Report No. UIUCDCS-R-80-1043 UILU-ENG 80 1744

SOME RESULTS ON THE WORKING SET ANOMALIES IN NUMERICAL PROGRAMS

by

Walid A. Abu-Sufah and David A. Padua

November 1980 NSF-OCA-MCS76-81686-000053

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. UIUCDCS-R-80-1043

SOME RESULTS ON THE WORKING SET ANOMALIES IN NUMERICAL PROGRAMS*

by

Walid A. Abu-Sufah and David A. Padua

November 1980

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the National Science Foundation under Grant No. US NSF MCS76-81686.

Abstract

This paper shows that the Working Set parameter-real memory and real memory-fault rate anomalies mentioned by Franklin, Graham, and Gupta in [FrGG78] do occur in traces generated by real programs. The results of the detailed investigation of this anomalous behavior in four Fortran programs are presented. In some cases, a drop of a factor of two in the average memory allotment is observed when the window size is increased. In some instances, a bigger memory allotment means an order of magnitude increase in page faults.

Keywords:

Memory Management
Multiprogramming
Working Set
Working Set Anomaly
Program Behavior

1. Introduction

Franklin, Graham, and Gupta have shown in [FrGG78] that the page fault frequency policy of memory management can exhibit anomalous behavior for some reference strings. They gave short example reference strings to illustrate their ideas and pointed out that real programs exhibit this anomalous behavior [Grah76], [Gupt74]. For the working set policy, WS, they gave an example reference string to demonstrate that certain anomalous behavior is also possible. However, nothing was mentioned about encountering these anomalies of the WS in real programs. The WS anomalies were encountered experimentally in the spring of 1978 by one of us while working on the development of automatic program transformations to improve program behavior in a virtual memory environment [AbuS78], [AbKL79], [AbKL80].

Before proceeding, we will describe the notation used in this paper, and will define the two types of anomalies to be considered. The WS policy keeps in memory the pages referenced during the previous τ memory references, where τ is the WS control parameter. This set of pages is the working set and is denoted by $W(\tau,t)$ at time t. Its size is $w(\tau,t)$. The average memory allocated to a program during its execution is given by

$$M(\tau,L) = \left(\sum_{t=1}^{R} w(\tau,t) + L \cdot \sum_{i=1}^{f(\tau)} w_i(\tau,t_i)\right)/(R + L \cdot f(\tau)) \qquad (1)$$

where

R = the length of the reference string

L = the mean page fault service time

$f(\tau)$ = the number of page faults

$w_i(\tau,t_i)$ = the working-set size at the ith page fault.

We say that there is a <u>parameter-real memory</u> anomaly when $M(\tau_1,L) > M(\tau_2,L)$ for some $\tau_1 < \tau_2$ and some L, and a <u>real memory-fault rate</u> anomaly when, for some $\tau_1 \neq \tau_2$ and some L, $M(\tau_1,L) < M(\tau_2,L)$ and $f(\tau_1) < f(\tau_2)$ [FrGG78].

In the 1978 experiments mentioned above, some of the transformed programs and some of the untransformed ones showed both anomalies. In those experiments only references to array elements were considered. The page size was 64 words and we used three values for L: 32, 320, and 3200 references. Table 1 summarizes our findings for these programs, and Table 2 shows some of their characteristics. In Table 1, we can see that the two anomalies defined above arise in all the programs listed for at least one value of L. Consider, for example, the program BASE when L = 32. When τ is 6, the average memory used is 2.33 pages and the number of page faults is 681. When τ is increased to 8, the average memory decreases to 1.96 pages and the number of page faults decreases drastically to 374. The decrease in average memory when τ increases illustrates the parameter-real memory anomaly, and the decrease in the number of page faults when the average memory decreases, illustrates the real memory-fault rate anomaly.

Recently, Denning in [Denn80] argues that it is unlikely that any nonlookahead policy better than the WS will be discovered. Thus, it seems that WS dispatchers should be widely used in future computer systems. However, load control of a multiprogrammed system which is based on an anomalous memory management policy may be unstable since a change

Table 1. Programs with Anomalous Behavior under WS

Program	τ_1	τ_2	$f(\tau_1)$	$f(\tau_2)$	$M(\tau_1,32)$	$M(\tau_2,32)$	$M(\tau_1,320)$	$M(\tau_2,320)$	$M(\tau_1,3200)$	$M(\tau_2,3200)$
BASE+	6	8	681	374	2.33	1.96	2.95	2.37	3.08	2.49
CD+	4	5	1904	92	2.79	2.73	2.89	2.82	2.92	3.00
FIELD+	6	7	1261	133	3.40	2.86	3.64	3.47	3.67	3.68
MAIN+	6	8	6024	1004	3.62	3.06	4.04	3.69	4.11	3.96
MAMOCO+	8	16	70938	1878	5.48	4.83	5.59	4.88	5.60	4.91
DISPERSE	464	528	948	814	29.37	28.66	32.06	29.65	32.53	29.84
FOURTR	320	384	2788	1903	38.15	37.05	38.62	34.45	38.71	33.84
INIT	5	6	1417	847	3.80	3.40	3.99	3.51	4.02	3.53
	448	512	437	245	32.38	26.55	38.20	26.72	39.28	26.76

+Transformed programs

Table 2. Some Characteristics of the Programs with Anomalous Behavior

Program	# of Arrays Referenced	# of Array References	# of Pages Referenced	Comments
BASE+	30	24 143	301	Cloud Physics Program from UIARL*
CD+	3	79 528	23	Cholesky Decomposition (48x48 Syst.)
FIELD+	20	11 152	52	Electromagnetic Fields Prg. from AFWL*
MAIN+	40	102 497	200	Cloud Physics Program from UIARL*
MAMOCO+	8	236 027	875	Matrix Mode Coupling Program from NRL*
DISPERSE	52	23 659	734	Chemical Analysis Program from NSF
FOURTR	7	86 012	128	Fast Fourier Transform Program (1024 points) from NRL*
INIT	25	12 154	245	Initialization Program from AFWL*

+Transformed programs

*UIARL - University of Illinois Atmospheric Research Laboratory

*AFWL - Air Force Weapons Laboratory

*NRL - Naval Research Laboratory

Table 3. Some Characteristics of the Programs Studied in Detail in This Paper

Program	Length of the Reference String		Number of Pages Referenced		Comments
	Array References	All References	Array Pages	All Pages	
BASE	21 747	174 384	251	256	See Table 2
FOURTR	86 012	752 922	176	183	See Table 2
INIT	12 154	147 112	195	202	See Table 2
PAPUAL	58 688	2150 347	520	526	Random Particle Velocity Generator Program from NRL*

*NRL - Naval Research Laboratory

of a given sign in the parameter might not produce changes of corresponding sign in the controlled variable [FrGG78]. This has convinced us of the importance of making a thorough investigation[1] and analysis of the WS anomalies in real programs which is the subject of this paper.

Specifically, we will study in detail four of the untransformed programs used in our previous experiments, namely: BASE, FOURTR, INIT, and PAPUAL. Table 3 shows some of their characteristics.[2] In our crude previous investigations, BASE and PAPUAL did not show any anomalies while INIT and FOURTR did. We start Section 2 of this paper with a brief discussion of the trace generation and processing method. Then we present the results of our experiments and their analysis. In Section 3, we make some concluding remarks.

---

[1]The investigation whose results are displayed in Table 1 was not thorough.

[2]The values in Table 3 assume column major storage for two-dimensional arrays, whereas those in Table 2 assume the square-block storage scheme

2. The Results and Their Analysis

2.1 The Experimentation Method

The page size in our experiments was 256 8-bit bytes. The main memory access time was taken as the time unit and the average access time to secondary memory was defined as L time units. The instruction set was assumed to be the IBM 370's.

We assumed the use of segments [Denn70], each consisting of one or more pages. For each program, one segment was allocated to the code, one to each array, and one to the scalar variables. Each variable was assumed to be four bytes long, and two-dimensional arrays to be stored in column-major order.

The tool we used to do the experiments is sketched in Fig. 1. It consists of two components; a trace generator and a simulator, both written in PL/I. The trace generator reads a source Fortran program, and the output from the Fortran G compiler for the same source program. The source program is interpreted to generate the address trace. The object program is used only to determine which and how many machine instructions are fetched for each Fortran statement. This approach to trace generation has some advantages. It allows us to generate subsets of the addresses in the trace; for example, we can generate array addresses only, or instruction addresses only. Also, the storage strategy can be changed without having to change the compiler; for example, two-dimensional arrays can be stored by rows, columns, or square blocks [McCo69]. For performance reasons, not all assignment statements in the source program are interpreted, only those that determine the address trace. For example, in the program

```
      DIMENSION  A(100), B(100)
      DO  1  I = 1, 100
1        A(I) = B(I) + C
      STOP
      END
```

the assignment statement does not have to be interpreted; the address trace will be the same whatever the value of A. The information on which assignment statements to interpret is supplied to the trace generator by the user.

The simulator of the WS policy is fairly straightforward; it takes a range of values and an increment for τ, and produces all the information required to compute the average memory for those values of τ. Since the average working-set size at fault time does not satisfy the inclusion property, we were not able to use a stack algorithm [MGST70]. This made the simulation costly in terms of computer time. In contrast, the trace generation was so cheap that we decided not to store the trace in a tape but to regenerate it each time a new simulation was done. The simulator was used as a subroutine of the trace generator.

2.2 The Results

2.2.1 The Page Faults vs. Window Size Curves

Fig. 2(a) shows the page faults vs. the window size, τ, curve for the array reference string of program BASE. Fig. 2(b) shows this curve when references to scalar variables, constants, and instructions are included in the reference string. Figs. 3, 4, and 5 show similar curves for the rest of the programs. We notice that the graphs for all the programs share some common characteristics.

The general shape of the page fault curves does not change when

references to scalar variables and instructions are included. This is obvious when comparing Figs. 2(a) and 2(b). Thus the paging behavior of our programs, and numerical programs in general, is mainly controlled by references to array variables (see also Figs. 6-9). This same argument was made in [MaBa76]. This is expected because the number of array pages referenced in numerical programs is much larger than the number of scalar and instruction pages (Table 3). However, there is around one order of magnitude difference between the total number of references in a string and the references to array elements. This explains the order of magnitude shift along the window size axis between the curves in Figs. 2(a) and 2(b).

The curves are not smooth curves. This can simply be explained by the fact that numerical programs mainly execute loops. Thus, when the window size is increased to the point where all the pages referenced in an iteration of a loop are included in the same working set, a sudden drop in the number of page faults will occur.

2.2.2 The Average Memory Allotment vs. Window Size and the Page Faults vs. Average Memory Curves

Figs. 6-9 show the average memory allotment vs. window size curves for $L = 0$ and $L \to \infty$.[3] The average memory allotment, at a given $\tau = \tau_1$, can be either an increasing or decreasing function of L. We have

$$M(\tau_1,L) = \left(\sum_{t=1}^{R} w(\tau_1,t) + L \cdot \sum_{i=1}^{f(\tau_1)} w_i(\tau_1,t_i)\right)/(R + L \cdot f(\tau_1))$$

$$= (A(\tau_1) + B(\tau_1) \cdot L)/(R + C(\tau_1) \cdot L) \qquad (2)$$

Thus,

$$\partial M(\tau_1,L)/\partial L = (B(\tau_1) \cdot R - C(\tau_1) \cdot A(\tau_1)/(R + C(\tau_1) \cdot L)^2$$

---

[3]The solid curves are for $L = 0$ and the dashed curves are for $L \to \infty$.

Therefore, $M(\tau_1,L)$ will always fall between $M(\tau_1,0) = A(\tau_1)/R$ and $M(\tau_1,\infty) = B(\tau_1)/C(\tau_1)$. Thus, the curves for $L = 0$ and $L \to \infty$ are envelopes to all other curves for $0 < L < \infty$. Fig. 10 shows $M(\tau,0)$, $M(\tau,20)$, $M(\tau,200)$, $M(\tau,2000)$, and $M(\tau,\infty)$ for program BASE (array references only). We observe that the curves for $M(\tau,2000)$ and $M(\tau,\infty)$ are fairly close to each other. This is also true for all the other programs. Thus, the $M(\tau,\infty)$ curve is a good approximation to the $M(\tau,L)$ curves when $L > 2000$.

Figs. 11-14 show the page faults vs. average memory allotment for the programs ($L = 0$ and $L \to \infty$) . From Figs. 6-14 , we see that the parameter-real memory, and the real memory-fault rate anomalies occur in all programs. Table 4 shows the points at which the anomalies are most significant.

Fig. 15 shows a typical section of the memory allotment curve where an anomalous behavior is seen. We note that the line $M = M_2$ intersects the curve at $\tau = \tau_a$ and $M = M_1$ at $\tau = \tau_b$. If we let $\tau_3 = \lfloor\tau_a\rfloor$ and $\tau_4 = \lceil\tau_b\rceil$, then we observe the following

(i) There will be anomalies between τ_1 and all τ's in the open interval (τ_1,τ_4)

(ii) There will be anomalies between τ_2 and all τ's in the open interval (τ_3, τ_2)

(iii) When generating the data for plotting the curve, if the increment in τ, $\Delta\tau$, was greater than or equal to $\tau_4 - \tau_3$, then the anomalous behavior in this section of the curve will not show up. We remark that if every

Table 4(a). The Worst Anomalies in the Programs (Array References)

Program	τ_1, τ_2	$f(\tau_1)$, $f(\tau_2)$	$M(\tau_1,20)$, $M(\tau_2,20)$	$M(\tau_1,200)$, $M(\tau_2,200)$	$M(\tau_1,2000)$, $M(\tau_2,2000)$	$M(\tau_1,\infty)$ $M(\tau_2,\infty)$
BASE	298,299	14936,257	230.4, 188.3	232.0, 140.2	232.2, 116.0	232.0, 112.0
FOURTR	90,95	11895,7658	35.9, 29.7	39.2, 31.5	39.7, 31.8	39.8, 31.8
	175,190	5314,3759	36.2, 32.6	39.2, 32.7	39.7, 32.7	39.8, 32.7
	345,380	2612,1832	39.2, 37.5	40.6, 34.9	41.0, 34.0	41.0, 33.9
	720,750	1245,880	44.3, 43.6	42.9, 38.8	42.3, 36.2	42.3, 35.8
	1370,1450	611,458	49.3, 49.4	46.0, 43.8	43.5, 38.3	43.1, 37.2
	2495,2505	366,325	54.6, 54.6	50.9, 50.2	46.6, 44.6	45.6, 43.2
INIT	20,25	2534,1022	13.7, 7.4	15.1, 7.4	15.3, 7.4	15.3, 7.4
	190,200	970,298	31.4, 20.6	37.0, 20.1	37.9, 19.9	38, 17.5
	495,500	296,200	28.0, 27.7	27.9, 26.6	27.9, 26.1	27.9, 26.0
PAPUAL	895,896	18662,521	410.0, 382.7	410.5, 304.0	410.6, 254.5	410.6, 245.9

Table 4(b). The Worst Anomalies in the Programs (All References)

Program	τ_1, τ_2	$f(\tau_1)$, $f(\tau_2)$	$M(\tau_1,20)$, $M(\tau_2,20)$	$M(\tau_1,200)$, $M(\tau_2,200)$	$M(\tau_1,2000)$, $M(\tau_2,2000)$	$M(\tau_1,\infty)$, $M(\tau_2,\infty)$
BASE	2375,2380	14951,272	225.6, 206.3	234.0, 186.1	235.3, 135.6	235.0, 112.0
FOURTR	645,674	13807,10070	29.4, 28.0	35.8, 31.6	38.1, 33.3	38.4, 33.6
	1201,1334	6982,4305	34.2, 33.7	39.1, 33.8	42.1, 33.9	42.6, 33.9
	2348,2596	3417,2112	39.4, 39.7	41.1, 38.2	43.0, 35.8	43.4, 35.0
	4645,5126	1668,1025	45.2, 45.8	45.0 44.2	44.8, 39.5	44.7, 37.1
	9463,10300	781,492	51.1, 51.7	50.2, 50.2	47.1, 44.0	45.1, 38.0
	20000,20150	414,381	56.9, 57.0	56.0, 56.0	51.4, 50.9	46.3, 44.7
INIT	180,185	2553,1041	10.3, 7.7	15.5, 7.2	17.5, 6.8	17.7, 6.7
	3755,3760	975,761	32.9, 32.5	37.5, 35.8	41.2, 39.0	41.9, 39.6
	3795,3800	761,331	32.7, 31.8	36.1, 30.5	39.3, 28.0	40.0, 27.1
	3875,3880	331,305	31.9, 31.9	30.6, 30.2	28.2, 26.7	27.2, 25.3
	3755,3920	975,303	32.9, 32.0	37.5, 30.2	41.2, 26.6	41.9, 25.2
PAPUAL	33100,33200	18668,527	414.2, 414.1	414.1, 407.3	414.0, 361.1	414.0, 251.3

possible anomalous section of the curve is to show, then $\Delta\tau$ must be taken to be 1.

Table 5 shows τ_3, τ_4, and $\Delta\tau$ for the anomalous regions of the average memory allotment curves for program INIT. We note that in the array reference string, no anomalies will be discovered if τ is always changed by increments greater than 946 references. For the mixed string, no anomalies will be discovered if the increments of τ are greater than 12375. We note that for this program the maximum anomalous change in the average memory allotment is 20.5 pages.[4]

Program FOURTR shows a more complicated anomalous behavior. We notice that there are overlapping anomalous regions. Thus, for array references, no anomalies will be discovered if τ is incremented by values greater than 2320 - 60 = 2260 (the largest τ_4 minus the smallest τ_3 of the regions). For the mixed reference string, this number is 17205 .

Each one of the eight traces considered in detail in this paper was studied for a limited range of the window size, as can be seen in the figures. In Table 6, we show the minimum space-time[5] product inside this range, and the window size where this minimum is reached. A lower bound on the space-time product for window sizes beyond the range studied is also shown. This lower bound was computed with the following formula.

$$LB(\tau^*,L) = ST(\tau^*,L) - (f(\tau^*) - w(R,R)) * w(R,R) * L$$

where τ^* is the largest value of the window size used in the experiments.

---

[4] These numbers and those presented in the rest of the discussion are for $L \to \infty$.

[5] When calculating the space-time cost, we use the expression

$$ST(\tau,L) = \sum_{t=1}^{R} w(t,\tau) + L \cdot \sum_{i=1}^{f(\tau)} w_i(t_i,\tau)$$

(footnote continued on page 16)

Table 5. Statistics of the Anomalous Sections of the Average Memory Allotment Curves for Program INIT

Array References				All References			
τ_1, τ_2	τ_3, τ_4	$\Delta\tau$	ΔM	τ_1, τ_2	τ_3, τ_4	$\Delta\tau$	ΔM
21,26	9,66	57	7.9	181,186	46,1291	1245	11
191,201	74,1020	946	20.5	3755,3920	2025,14400	12375	16.7
496,501	431,586	155	1.9				

Table 6. Relative Position of Anomalies and the Minimum Space-Time Product

		Minimum ST Product in Measured Range		ST Product Lower Bound Beyond Measured Range	Anomalies Detected	
		$\hat{\tau}$	$ST(\hat{\tau},2000)$		for $\tau < \hat{\tau}$	for $\tau \geq \hat{\tau}$
BASE	(Array Refs.)	1	41.8×10^6	67.5×10^6	N	Y
	(All Refs.)	2380	97.4×10^6	100.6×10^6	N	Y
FOURTR	(Array Refs.)	3000	31.2×10^6	23.6×10^6	Y	N
	(All Refs.)	27500	75.4×10^6	32.4×10^6	Y	N
INIT	(Array Refs.)	201	10.6×10^6	18.3×10^6	Y	Y
	(All Refs.)	186	15.2×10^6	29.2×10^6	Y	Y
PAPUAL	(Array Refs.)	1	67.0×10^6	287.4×10^6	N	Y
	(All Refs.)	46	503.2×10^6	1158.2×10^6	N	Y

[5](Continuation)

The first sum in the above equation is equal to $R \cdot M(\tau,0)$. The second sum is equal to $f(\tau) \cdot M(\tau,\infty)$. In the literature, [DKLP76], [Denn78], [GrDe77], the second sum is sometimes approximated by $f(\tau)\ M(\tau,0)$. Thus, the approximate expression for ST is given by

$$\hat{ST}(\tau,L) = R \cdot M(\tau,0) + L \cdot \frac{f(\tau)}{R}\ M(\tau,0) \cdot R$$

$$= R \cdot M(\tau,0)(1 + L \cdot f(\tau)/R) .$$

While $\hat{ST}(\tau,L)$ can be easily calculated for all τ from statistics generated after one scan of a reference string, this is not possible for $ST(\tau,L)$. Thus, calculating ST is much more expensive than $\hat{ST}$. Graham reports that this approximation can be in error by as much as 20% [Grah76]. For our programs, we found that the error can be as high as 70%. Figs. 16 through 19 show the relative error curves for our programs ($(ST - \hat{ST})/ST$ vs. τ).

When $\hat{ST}$ is used to approximate ST, then the VMIN algorithm [PrFa76] is usually used as the optimal algorithm to minimize space-time product. The accurate algorithm for minimizing space-time product was developed in [BDMS80]. In [BuDa80], it is shown that DMIN outperforms VMIN. They also show that the WS outperforms VMIN in some instances.

It is easy to see that for $\tau \geq \tau^*$, $ST(\tau,L) \geq LB(\tau^*,L)$. From the previous information, we can see that anomalies occur for window sizes both smaller and larger than those where the minimum space-time product occur.

3. Conclusion

The results of the experiments reported in this paper show that the anomalous behavior of the WS policy is not insignificant. A change in the window size of a given sign can cause more than 200% change in the average real memory allotted to a program in the unexpected direction and a corresponding change in the page fault rate of one order of magnitude (see Table 4). Thus, this anomalous behavior of the WS policy cannot just simply be ignored.

In real computer systems, people are interested in the turnaround time, throughput, and multiprogramming degree. The turnaround time of a job is related to its CPU execution time and paging rate. The window size of the WS policy controls the paging rate with no problems. In other words, the WS policy does not have the parameter-fault rate anomaly. Thus, the turnaround time of a job can be safely controlled under the WS policy.

The throughput of a system, however, is dependent on its multiprogramming degree [Denn78], which itself depends on the average real memory allotted to programs during their execution. Thus, the results in this paper cast some doubts on the reliability of the window size as a control of the multiprogramming degree. However, the limited scope of this work precludes making any final statement on this subject. More experiments, including experiments with nonnumerical programs, are required before such a final statement can be made.

Acknowledgments

We would like to thank Professor D. J. Kuck for his encouragement and support provided to this study. The excellent job done by Mrs. Vivian Alsip in typing this manuscript is gratefully acknowledged.

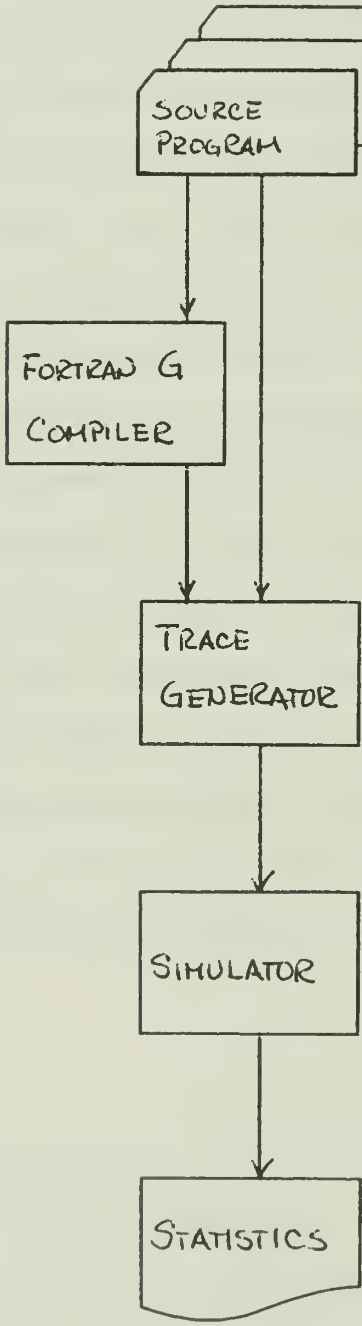

Fig. 1. Trace Generation and Simulation Facility

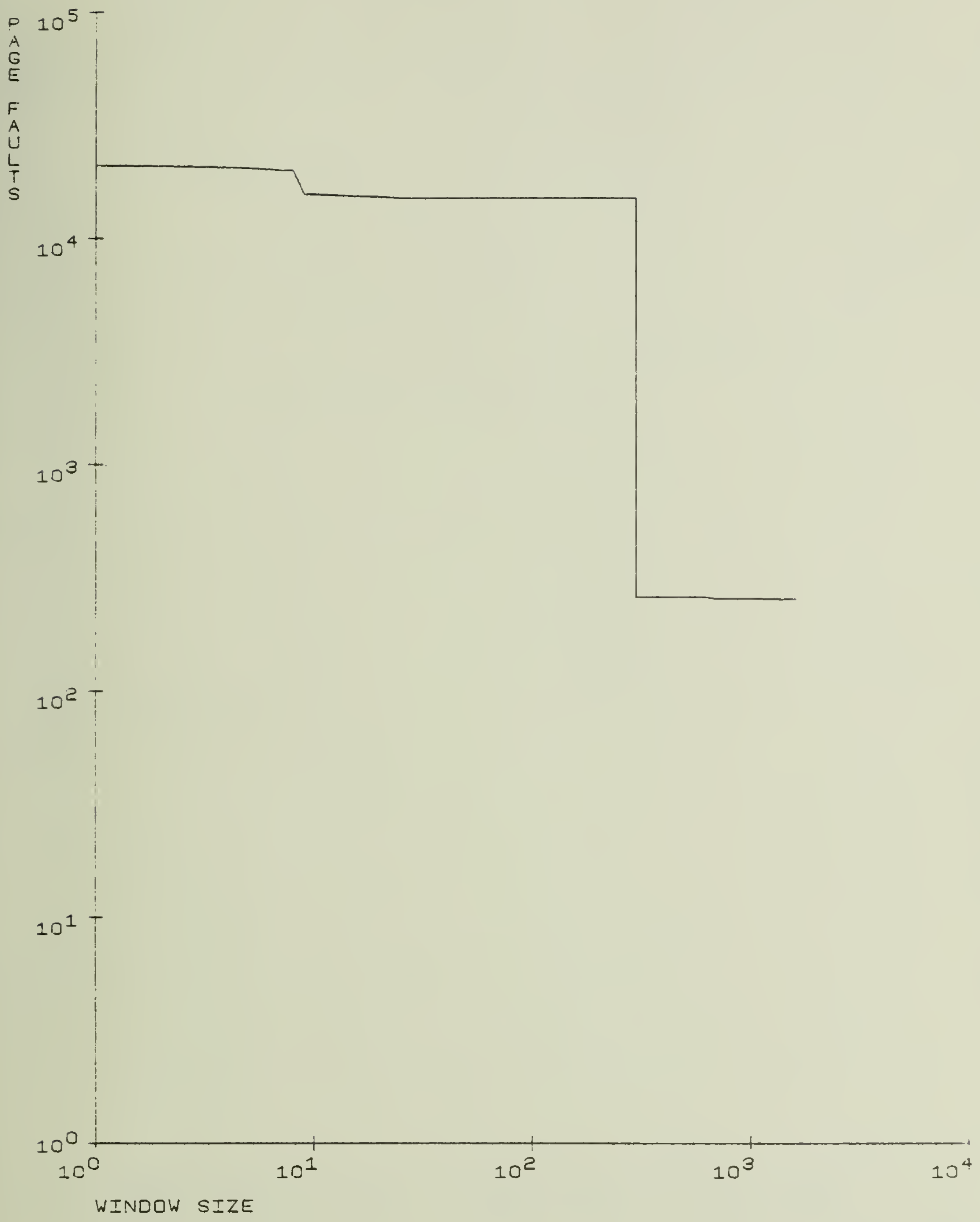

Fig. 2(a). The Page Fault Curve for Program BASE (Array References)

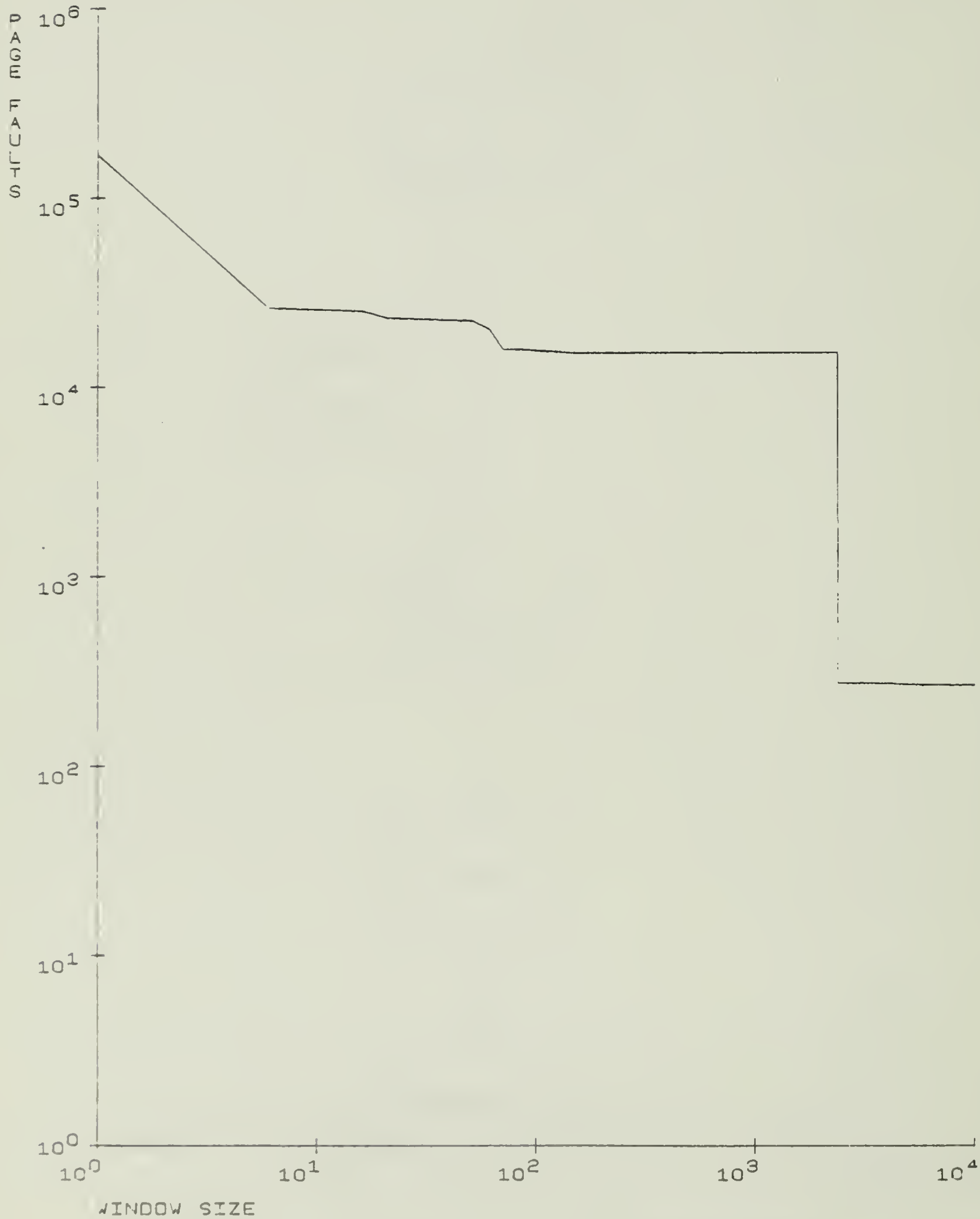

Fig. 2(b). The Page Fault Curve for Program BASE (All References)

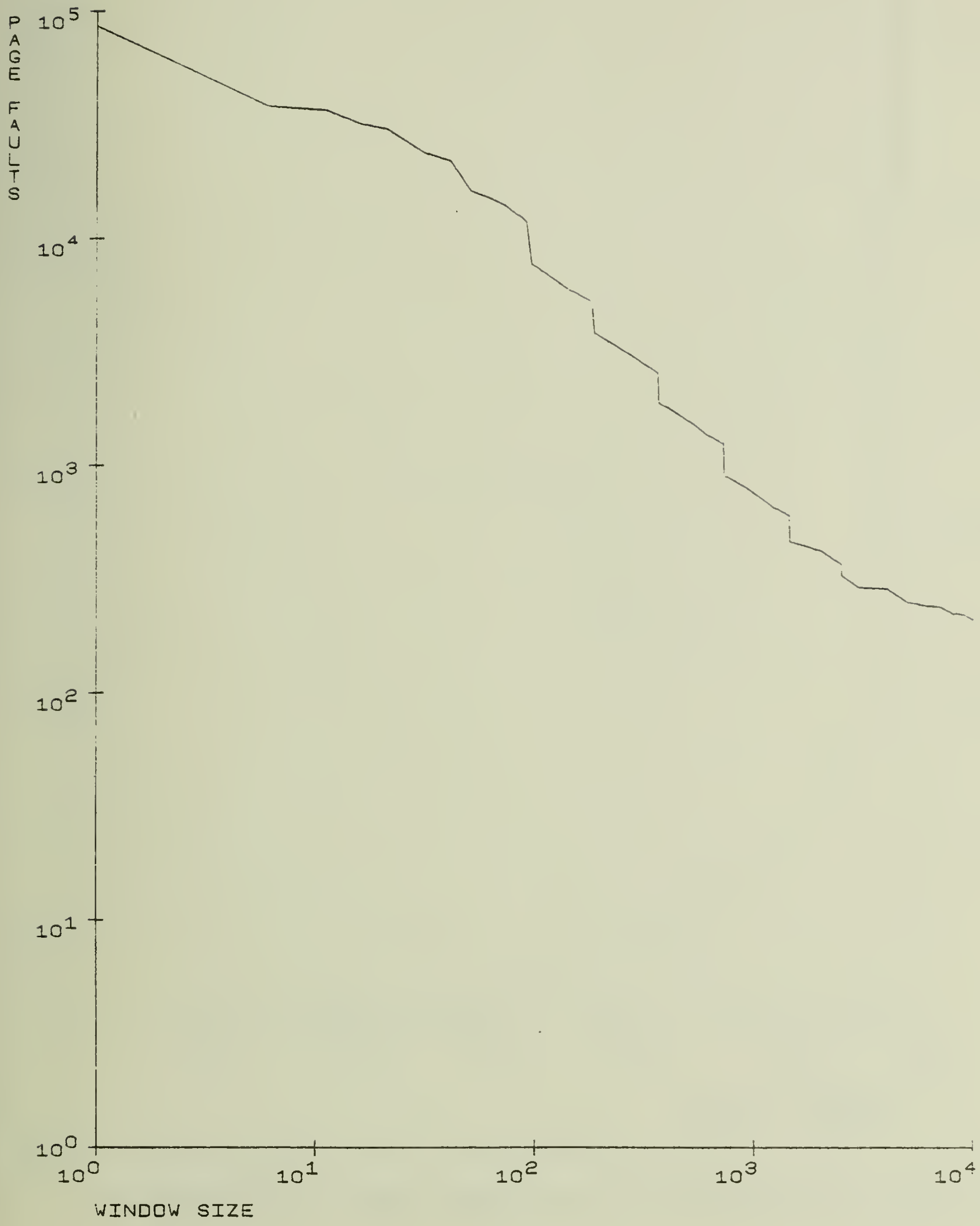

Fig. 3(a). The Page Fault Curve for Program FOURTR (Array References)

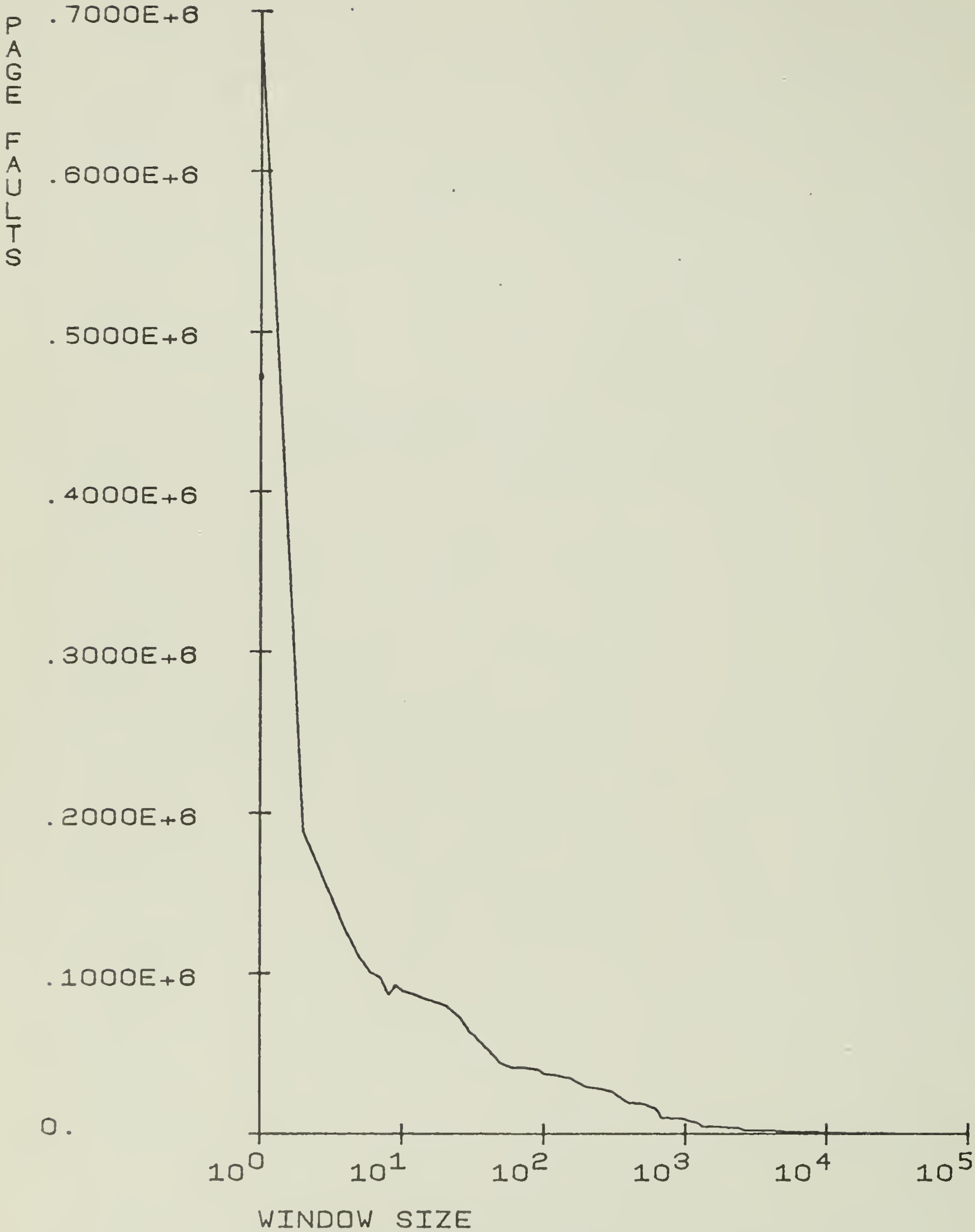

Fig. 3(b). The Page Fault Curve for Program FOURTR (All References)

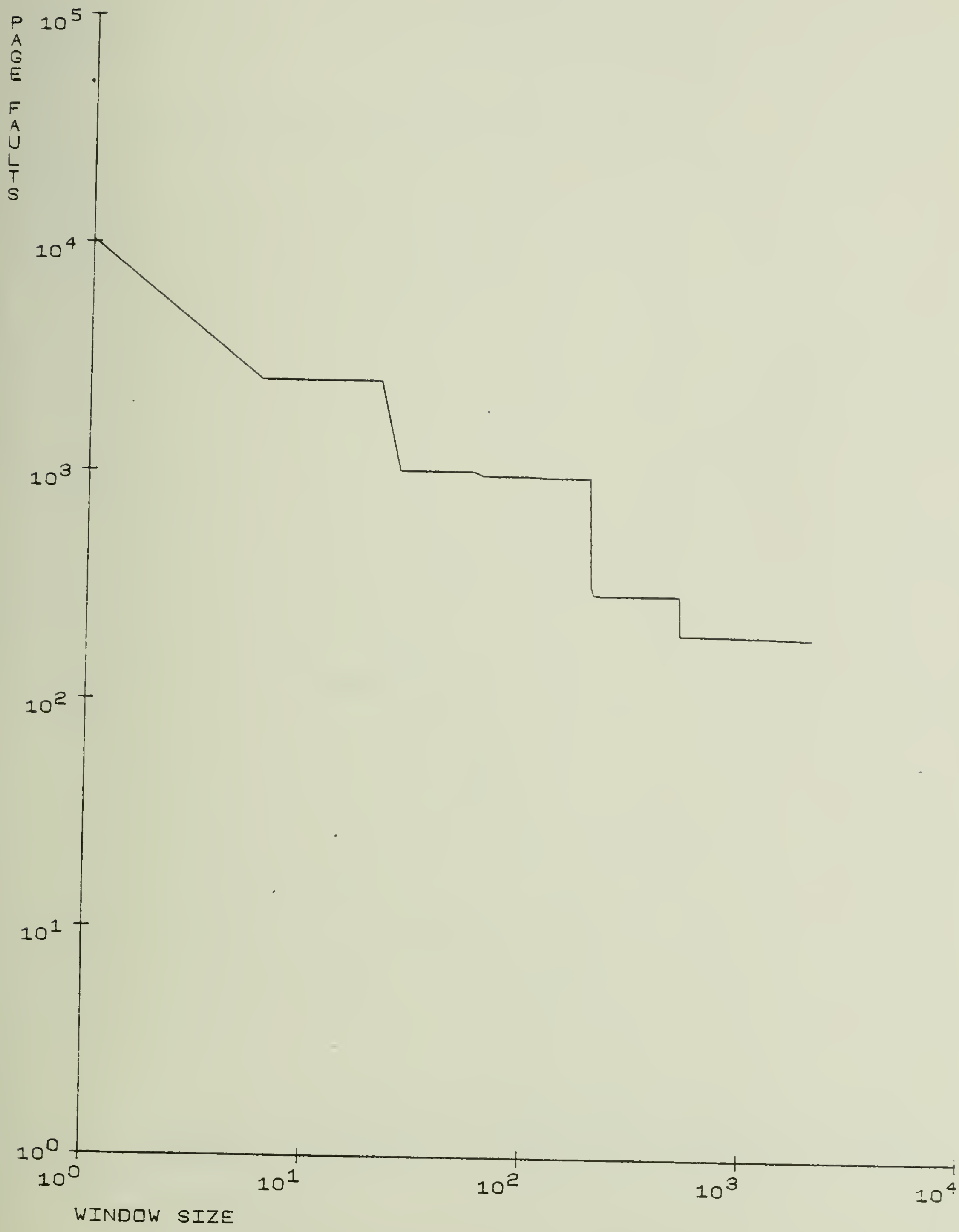

Fig. 4(a). The Page Fault Curve for Program INIT (Array References)

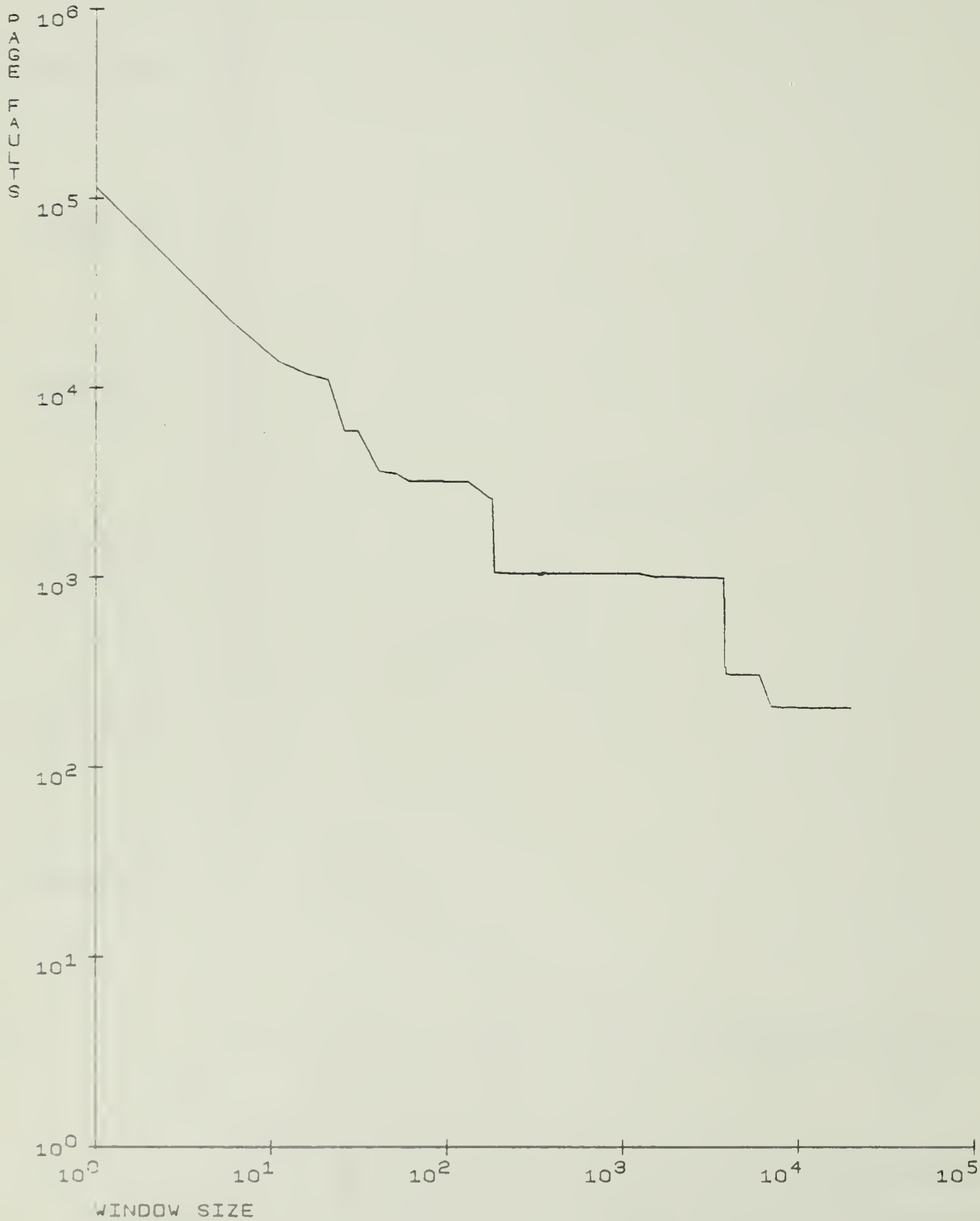

Fig. 4(b). The Page Fault Curve for Program INIT (All References)

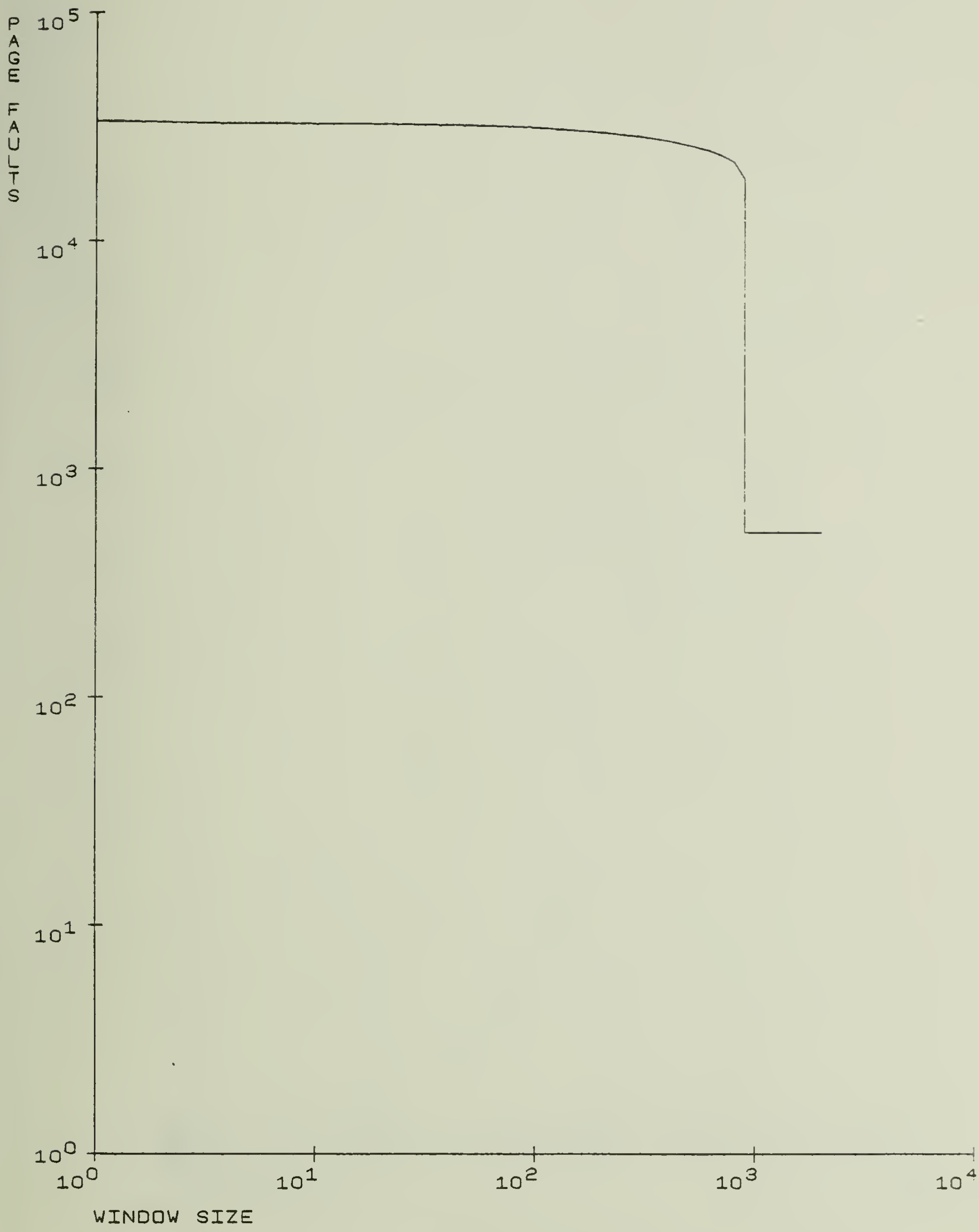

Fig. 5(a). The Page Fault Curve for Program PAPUAL (Array References)

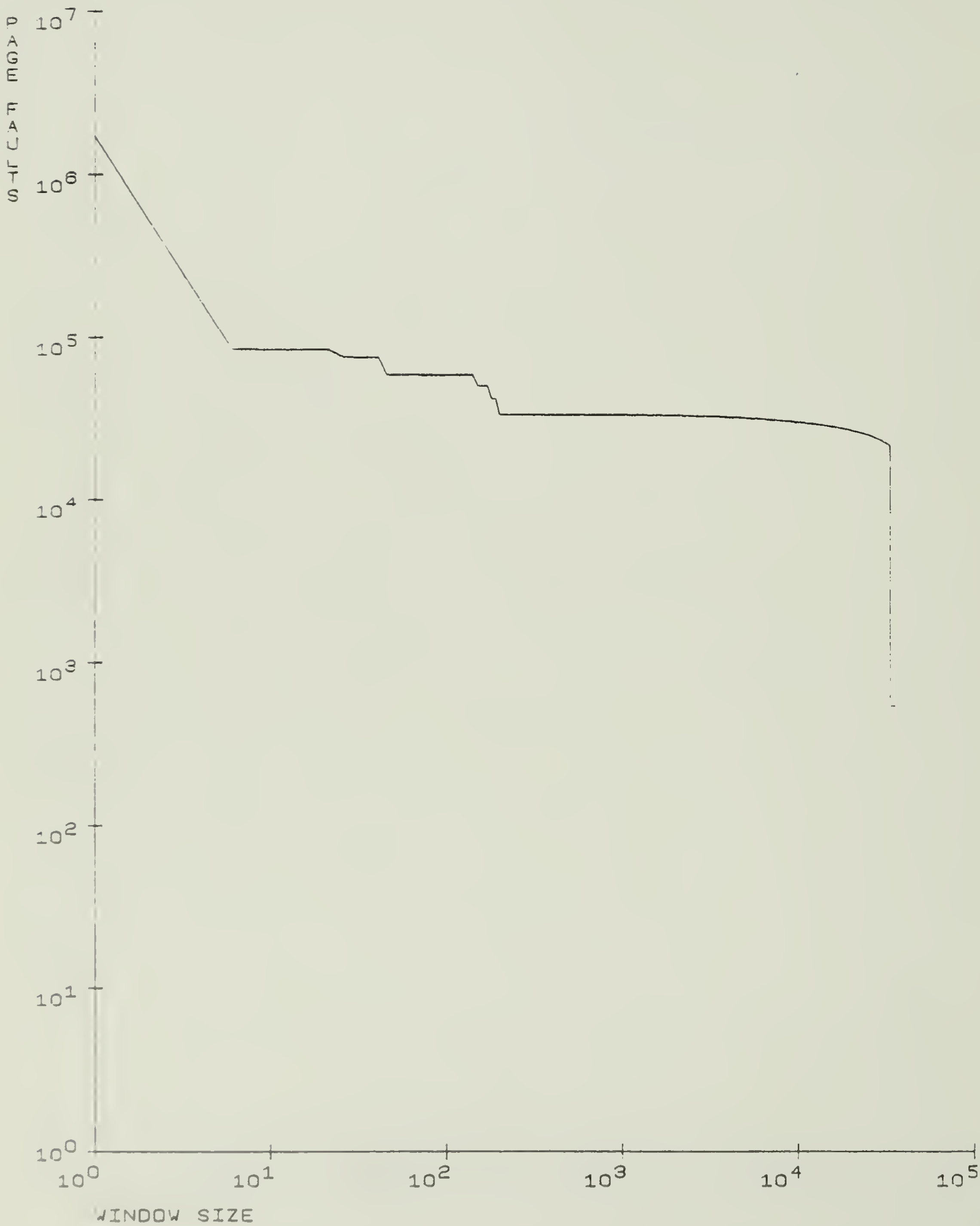

Fig. 5(b). The Page Fault Curve for Program PAPUAL (All References)

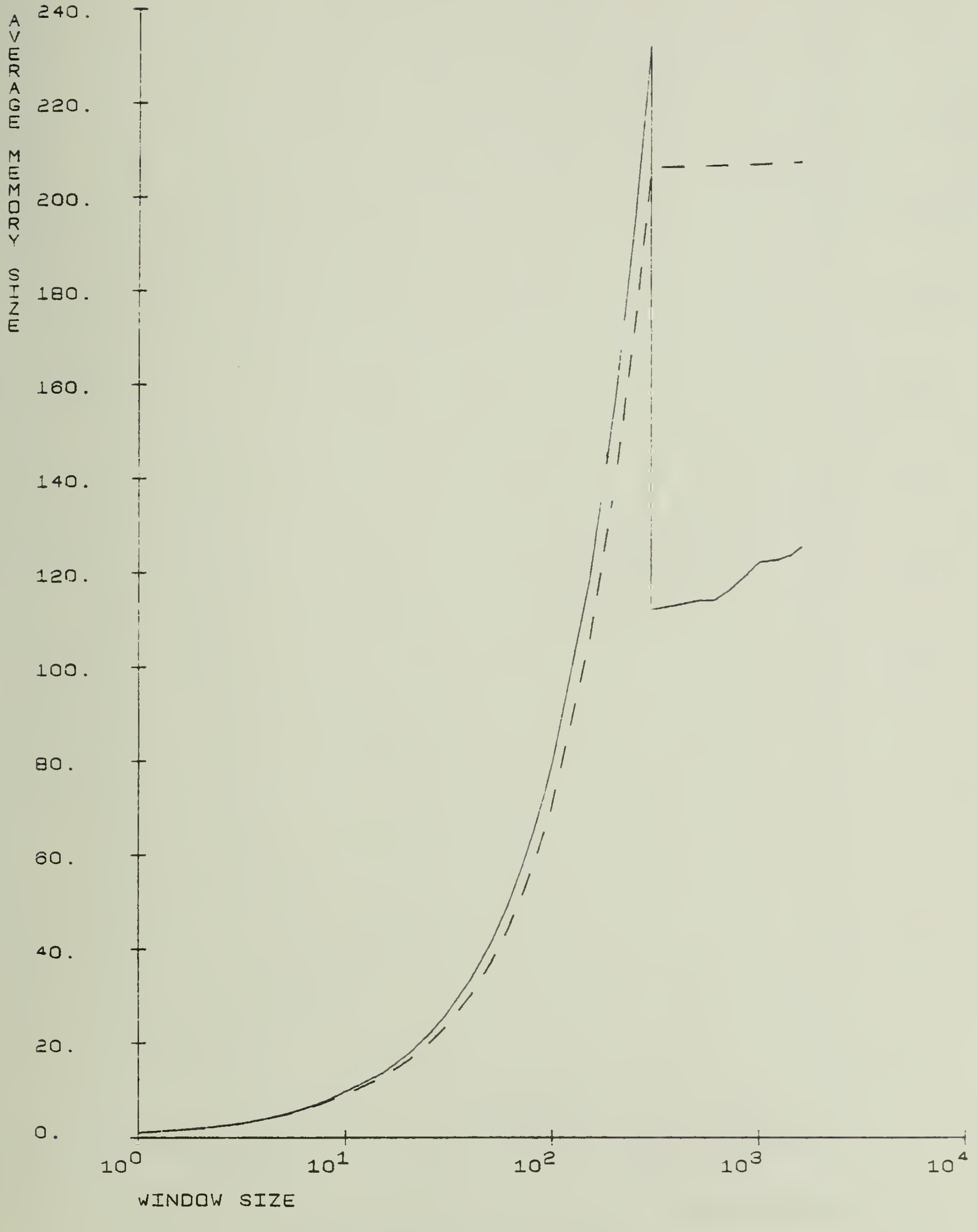

Fig. 6(a). The Average Memory Allocation Curves for Program BASE (Array References)

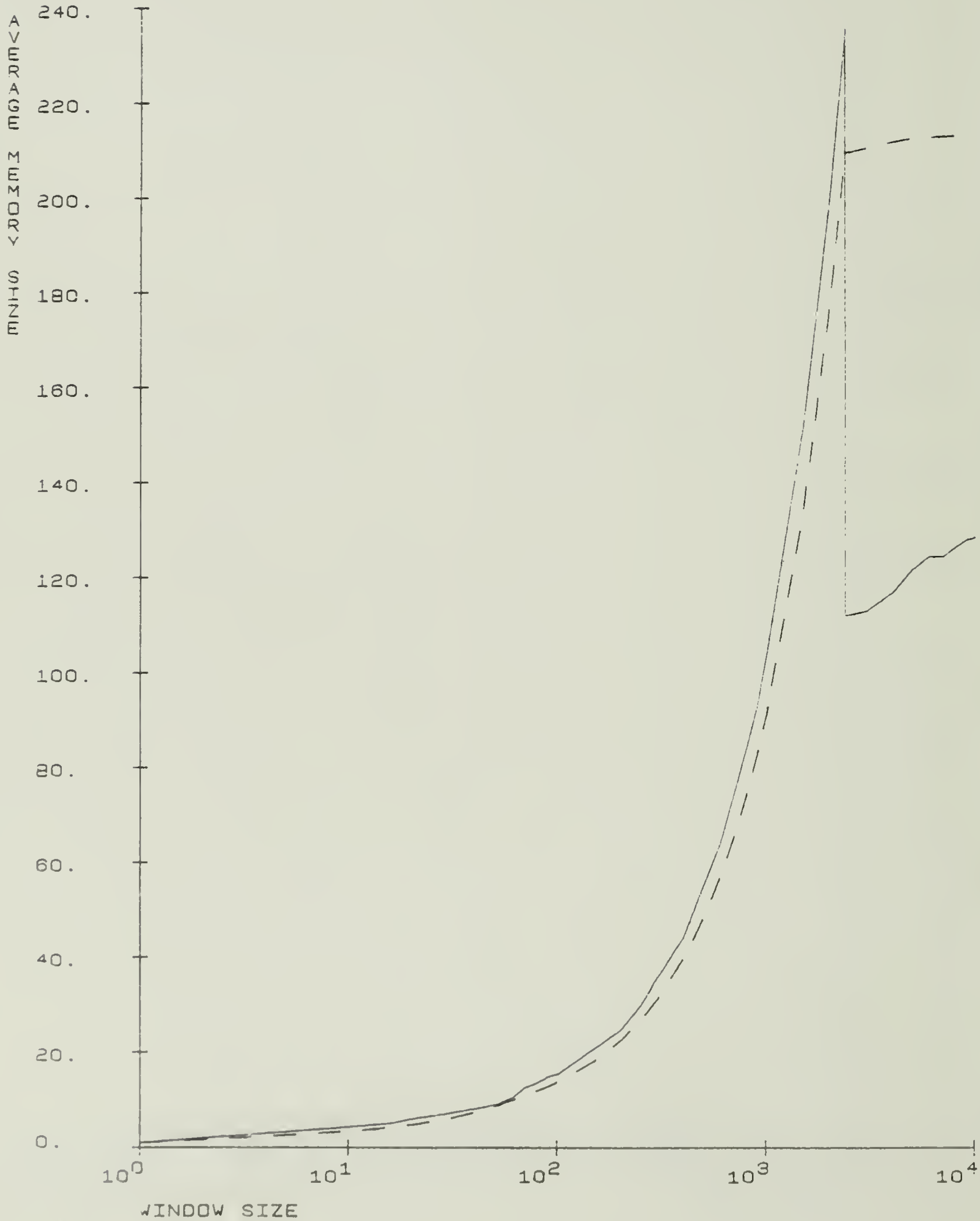

Fig. 6(b). The Average Memory Allocation Curves for Program BASE (All References)

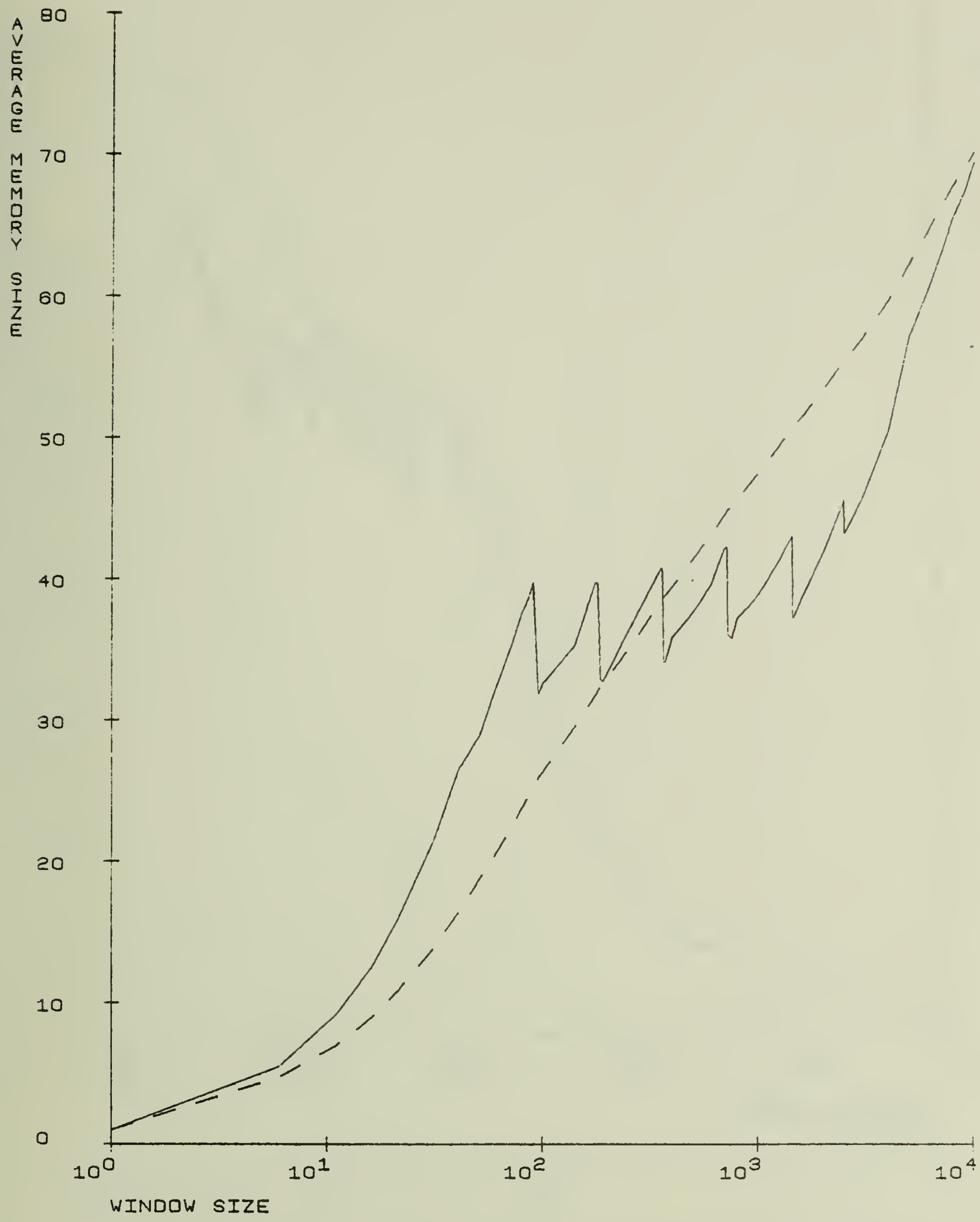

Fig. 7(a). The Average Memory Allocation Curves for Program FOURTR (Array References)

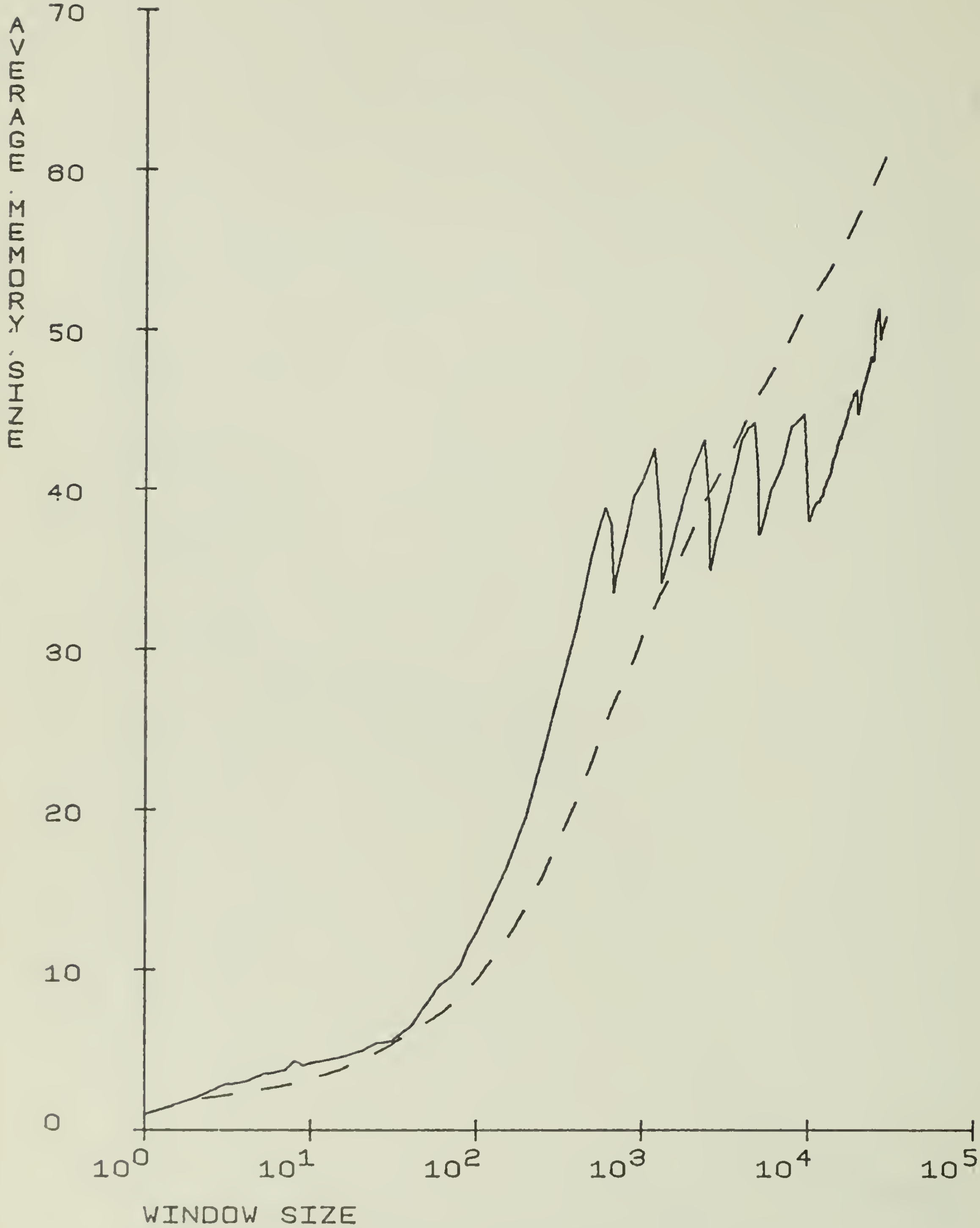

Fig. 7(b). The Average Memory Allocation Curves for Program FOURTR (All References)

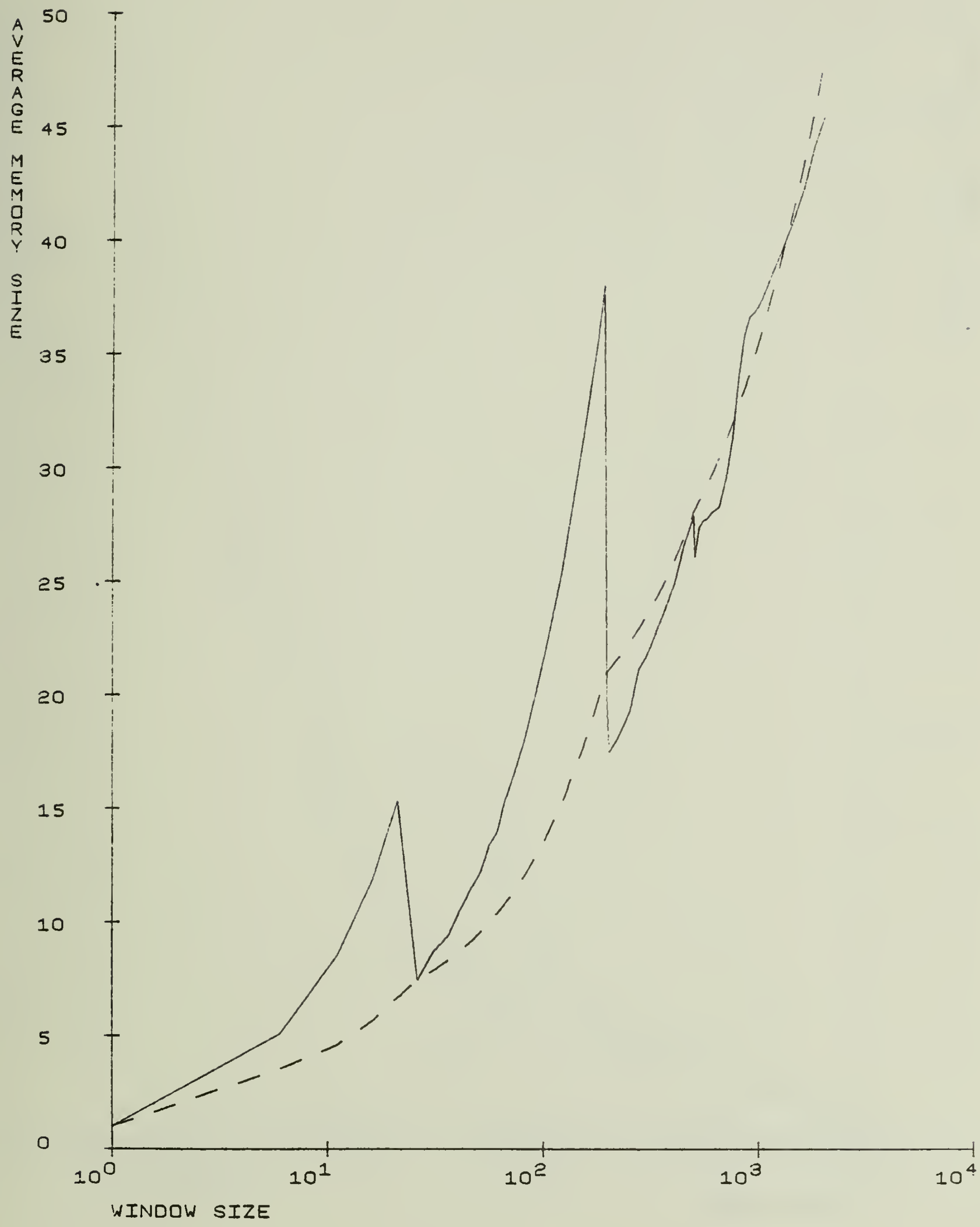

Fig. 8(a). The Average Memory Allocation Curves for Program INIT (Array References)

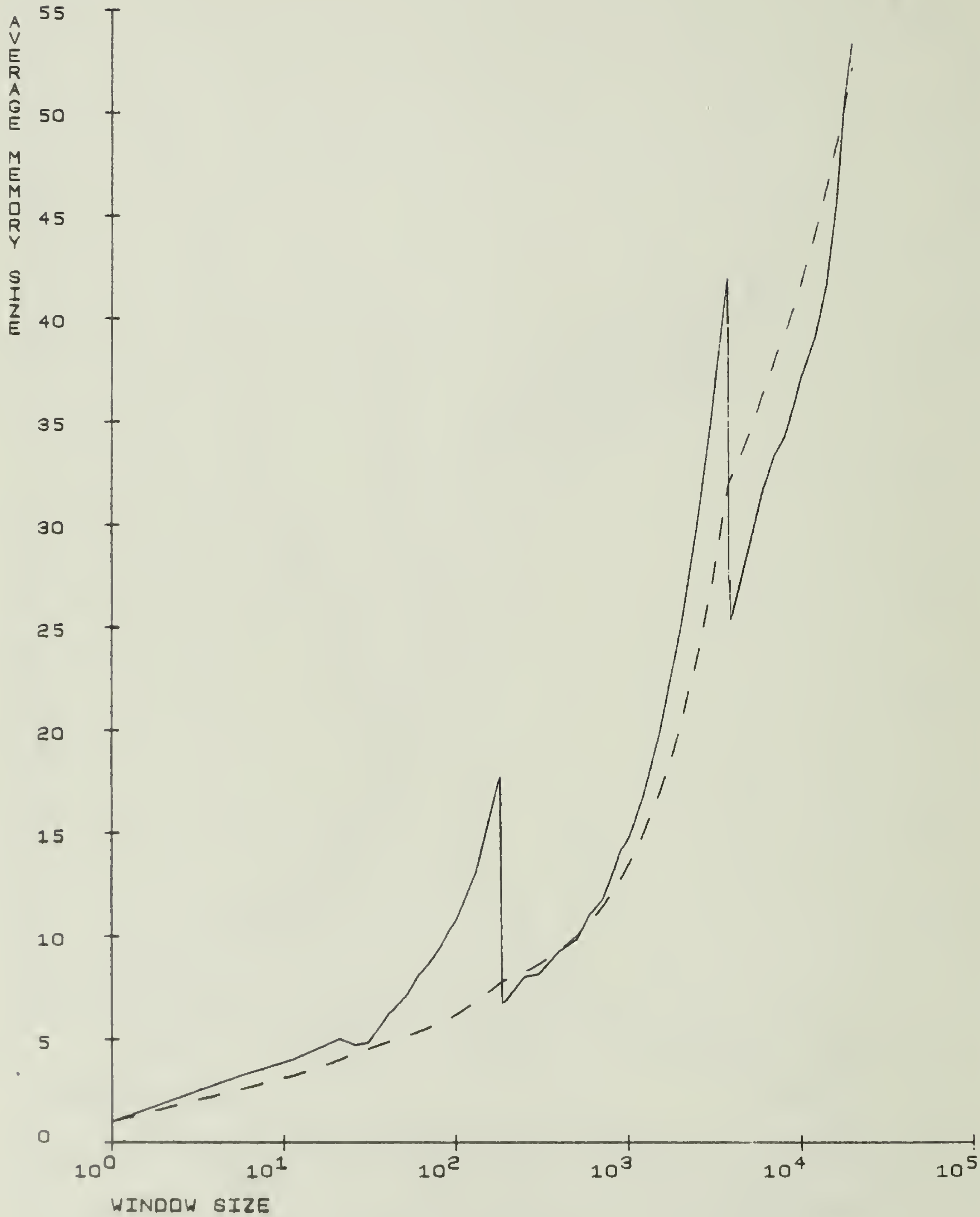

Fig. 8(b). The Average Memory Allocation Curves for Program INIT (All References)

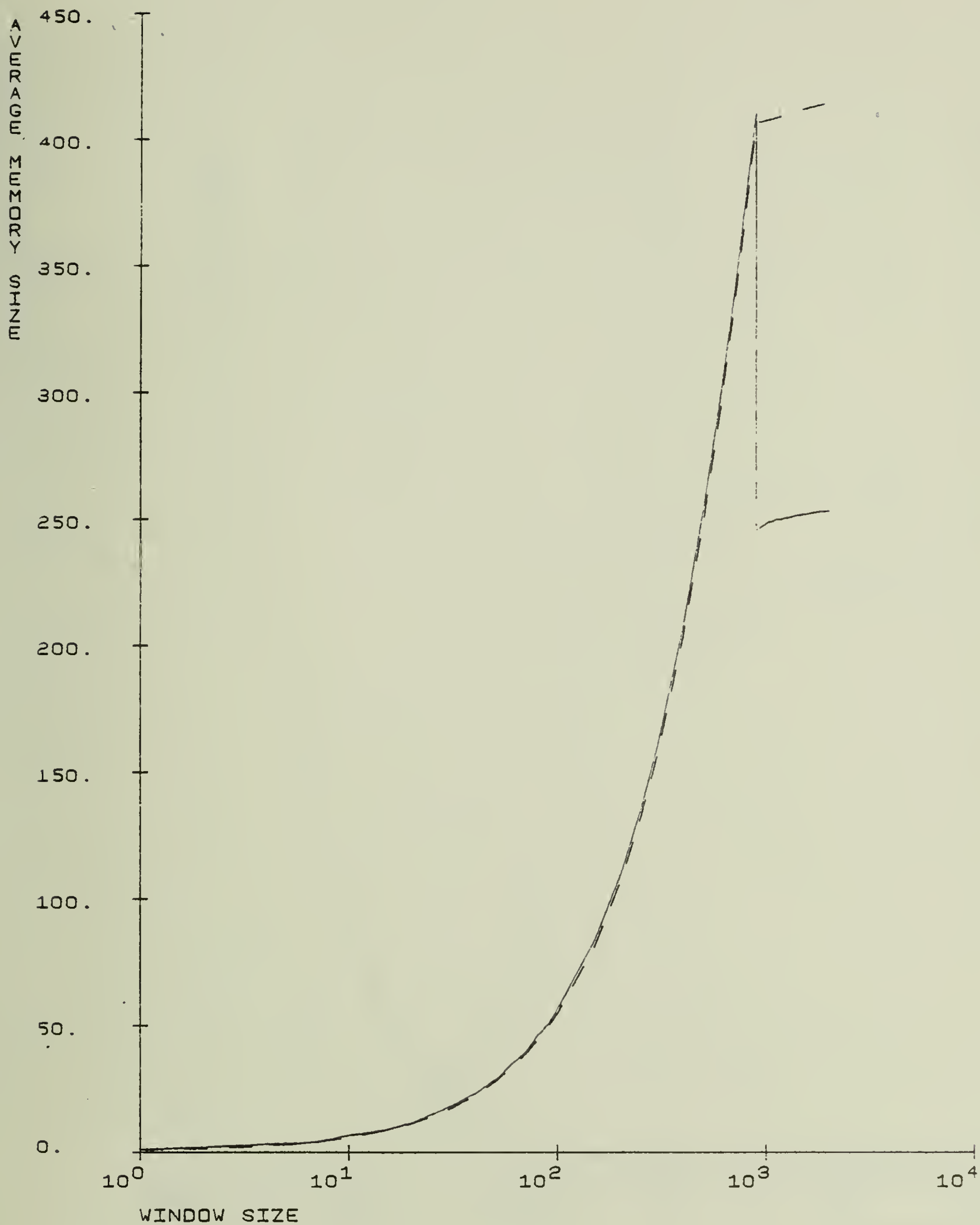

Fig. 9(a). The Average Memory Allocation Curves for Program PAPUAL (Array References)

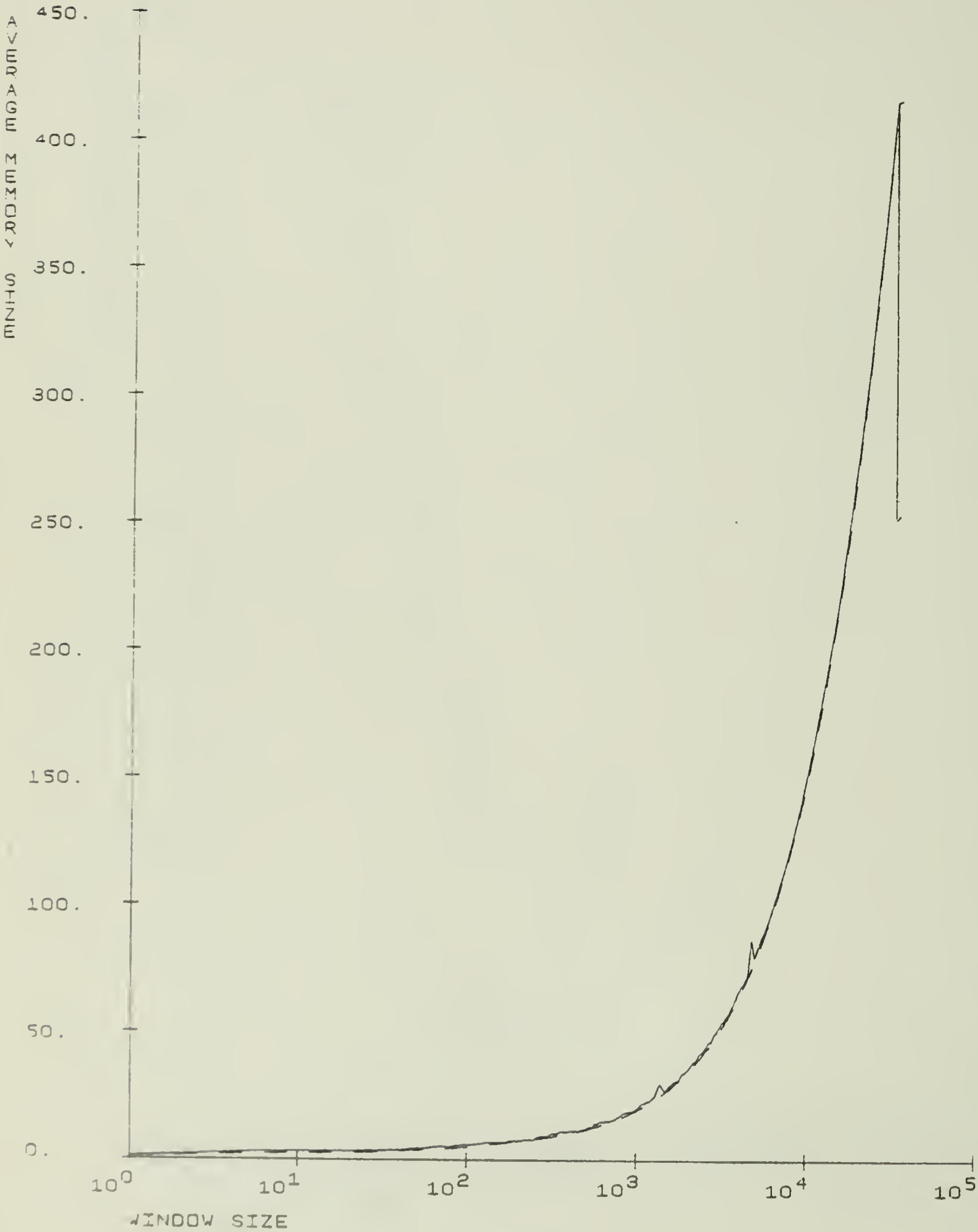

Fig. 9(b). The Average Memory Allocation Curves for Program PAPUAL (All References)

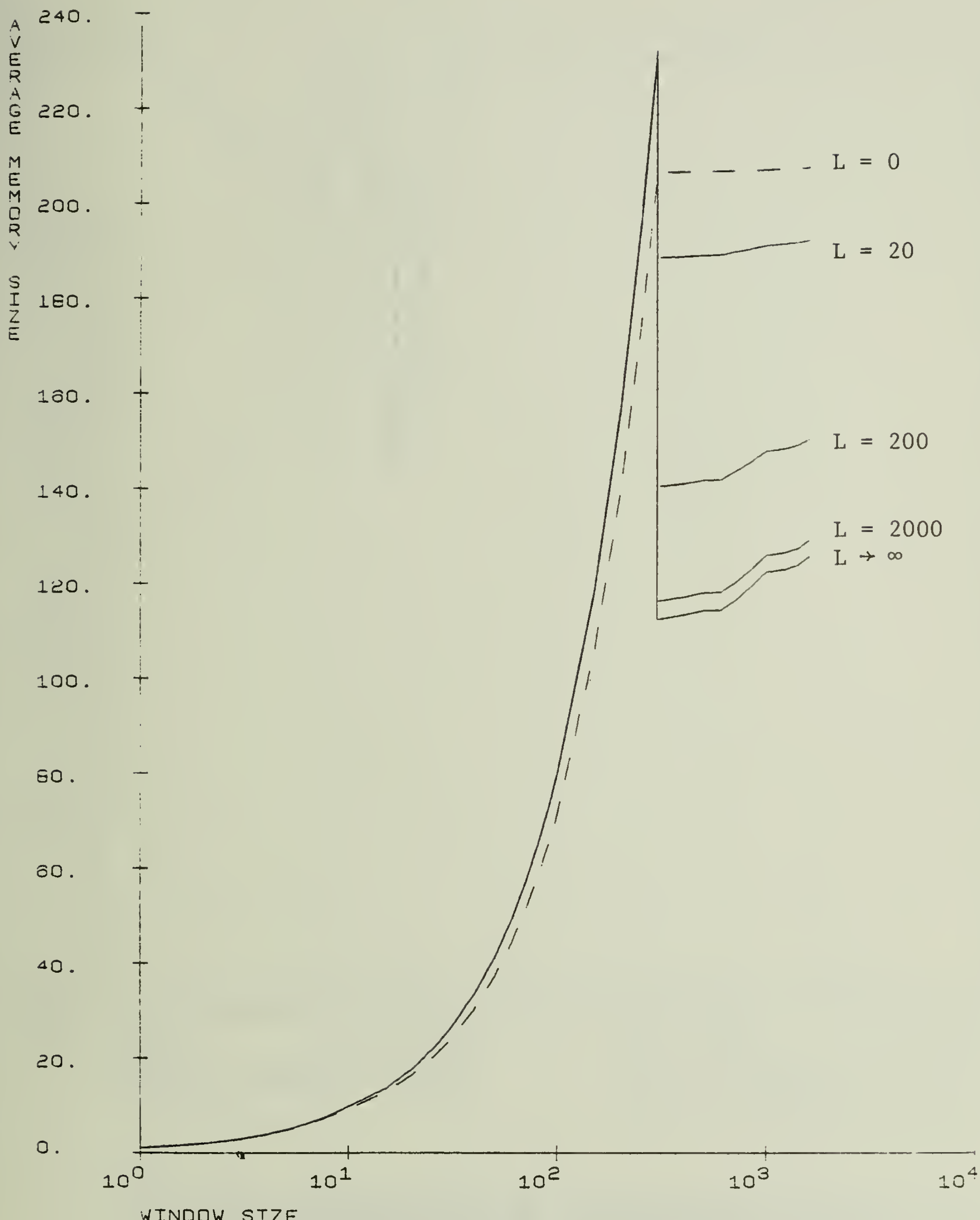

Fig. 10. The Average Memory Allotment Curves for Program BASE for Different Values of Page Fault Service Time

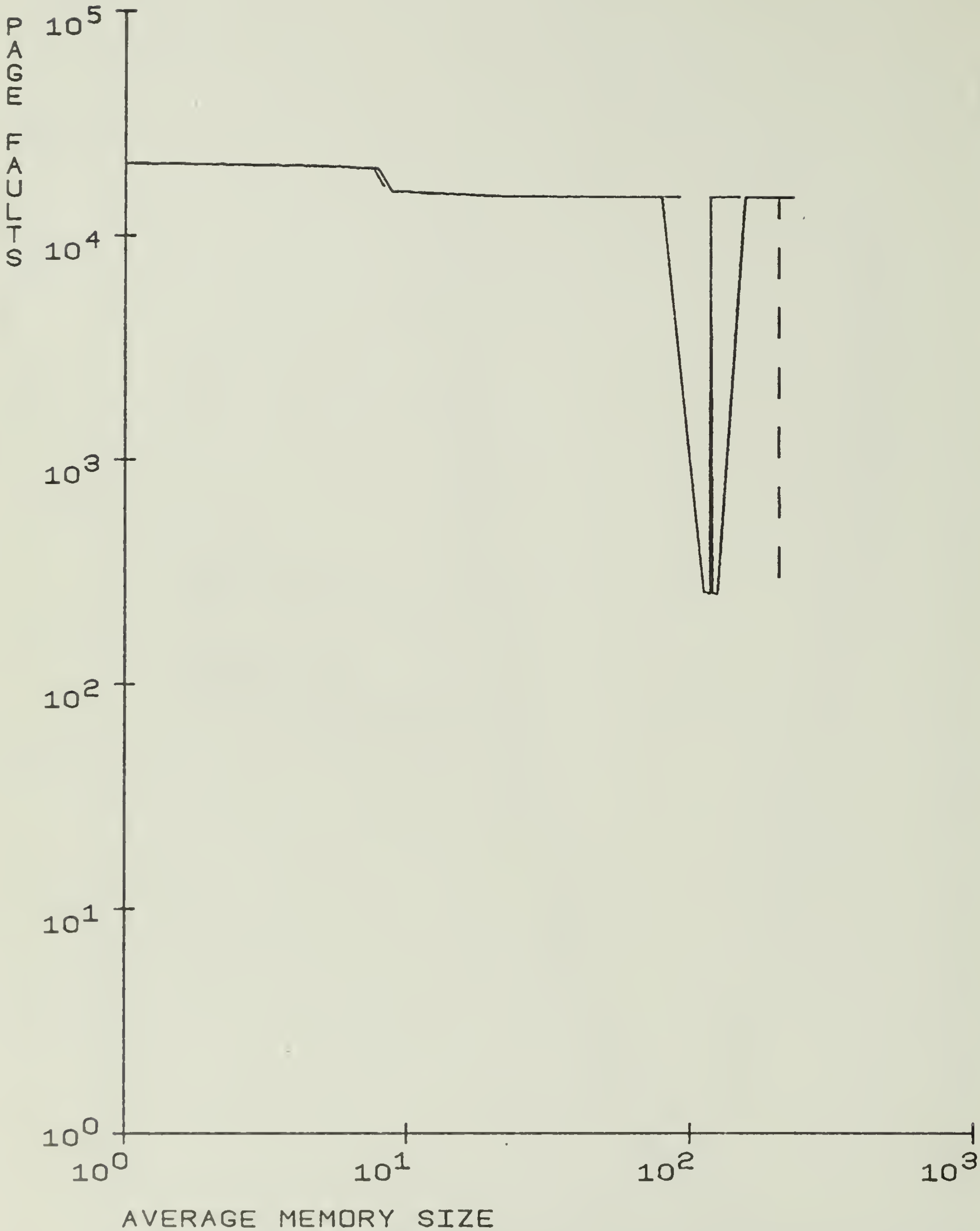

Fig. 11(a). The Page Faults vs. Average Memory Curves for Program BASE (Array References)

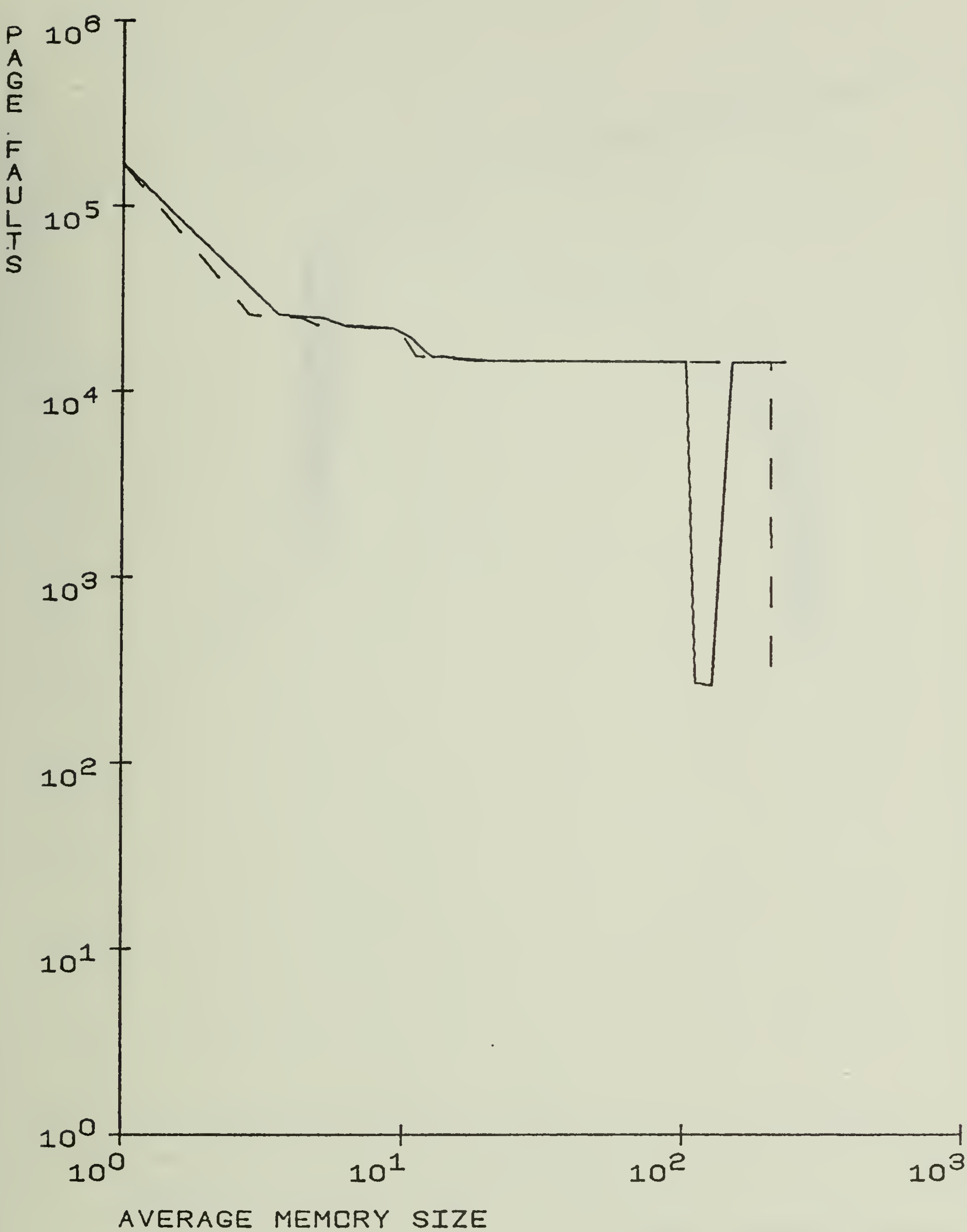

Fig. 11(b). The Page Faults vs. Average Memory Curves for Program BASE (All References)

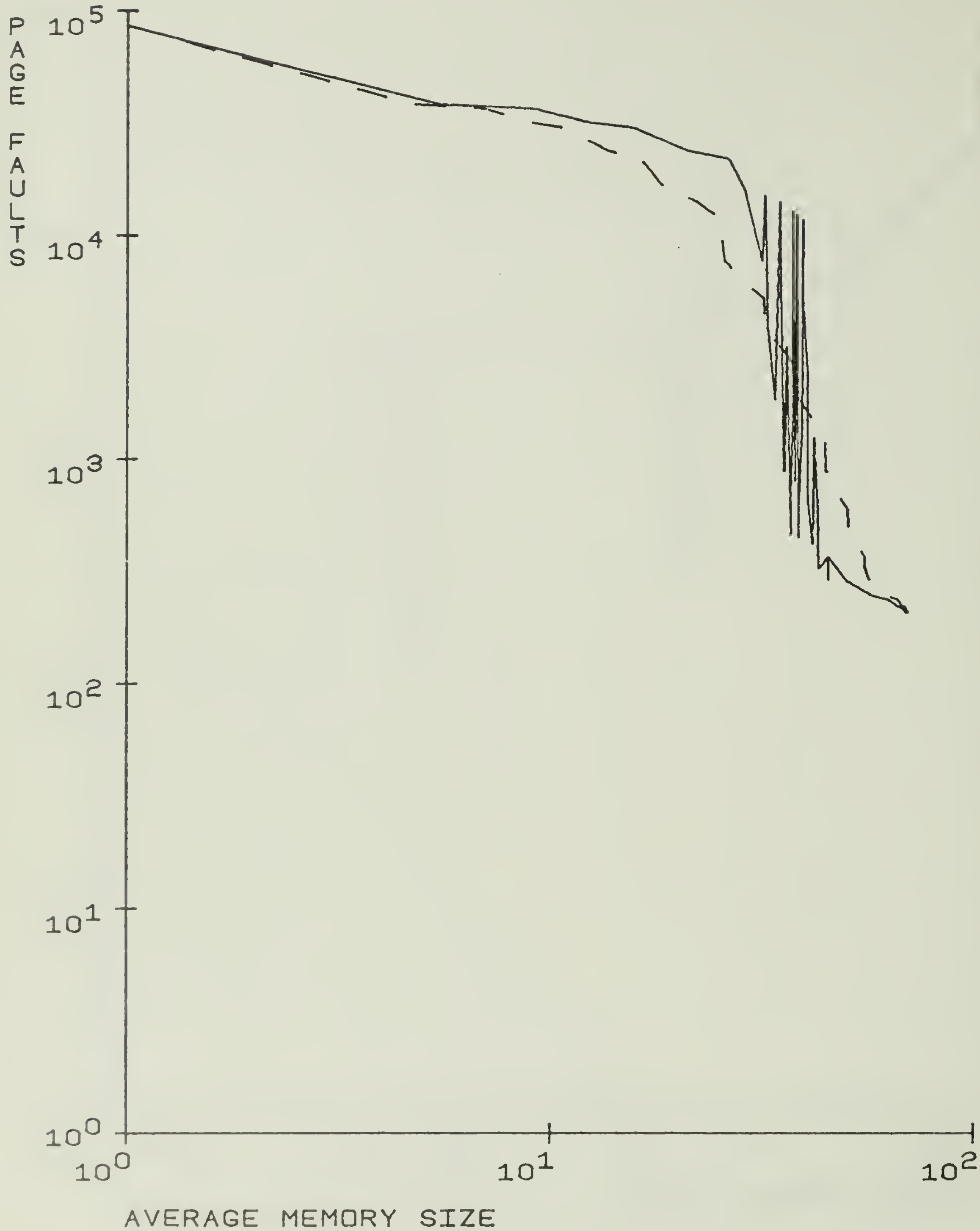

Fig. 12(a). The Page Faults vs. Average Memory Curves for Program FOURTR (Array References)

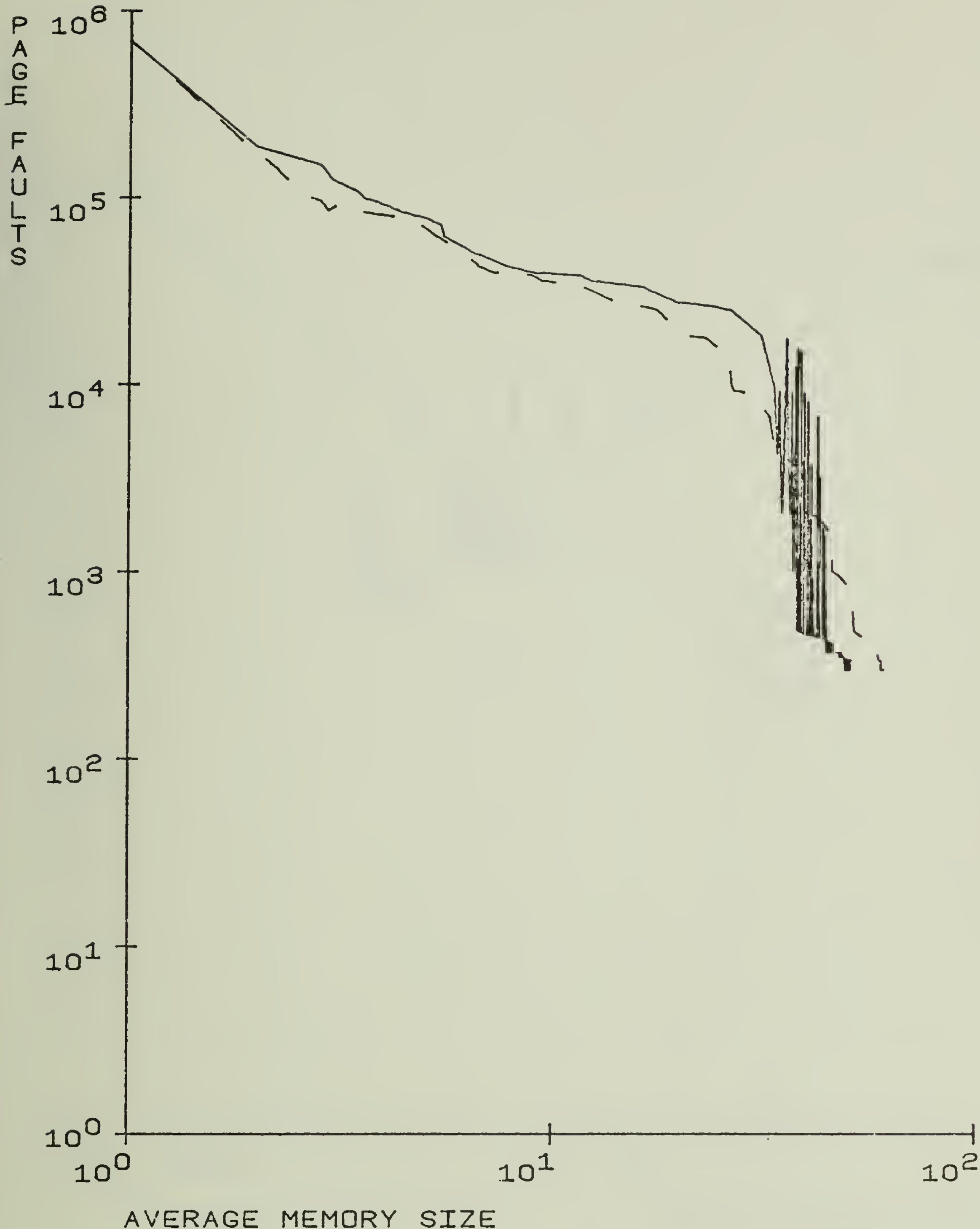

Fig. 12(b). The Page Faults vs. Average Memory Curves for Program FOURTR (All References)

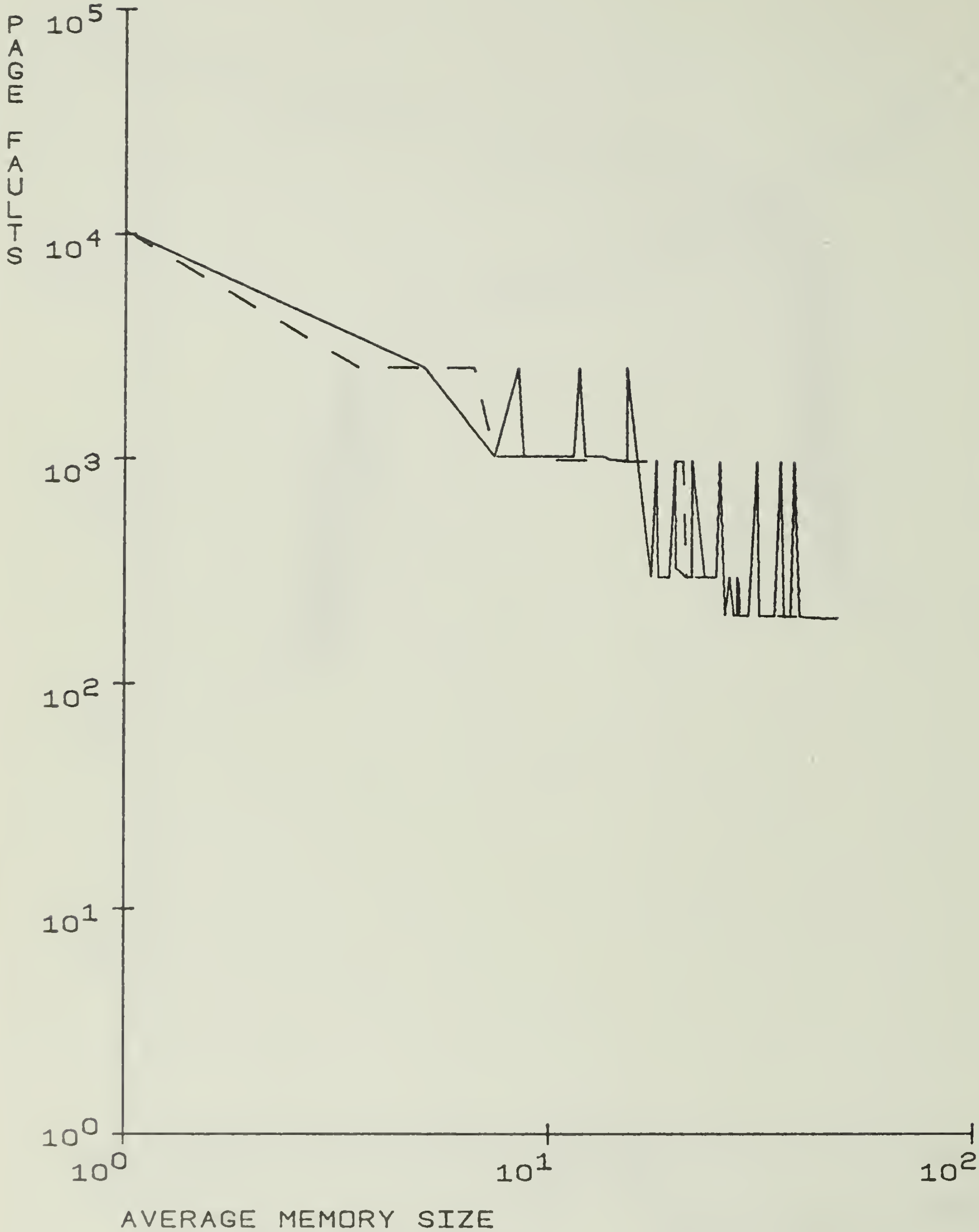

Fig. 13(a). The Page Faults vs. Average Memory Curves for Program INIT (Array References)

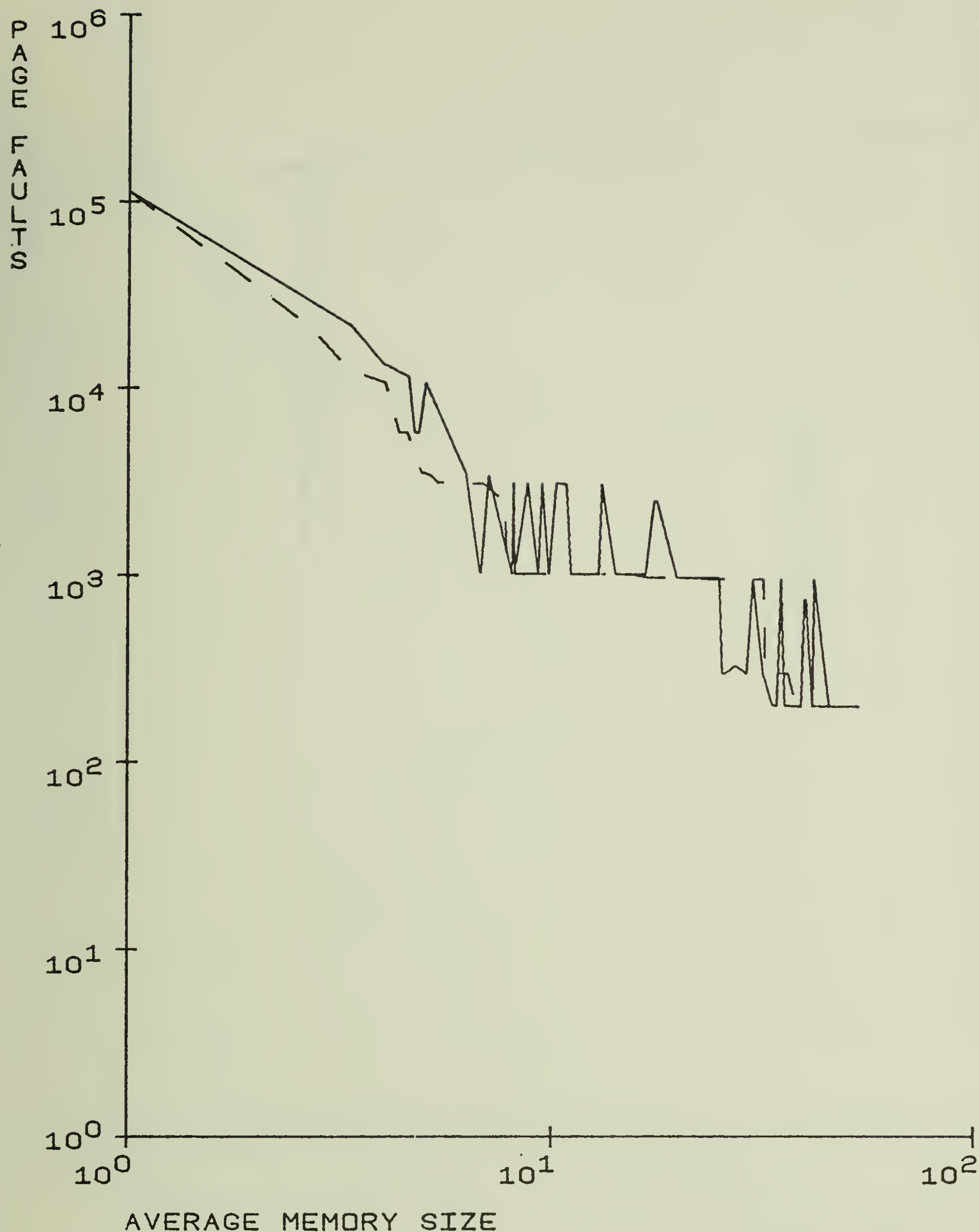

Fig. 13(b). The Page Faults vs. Average Memory Curves for Program INIT (All References)

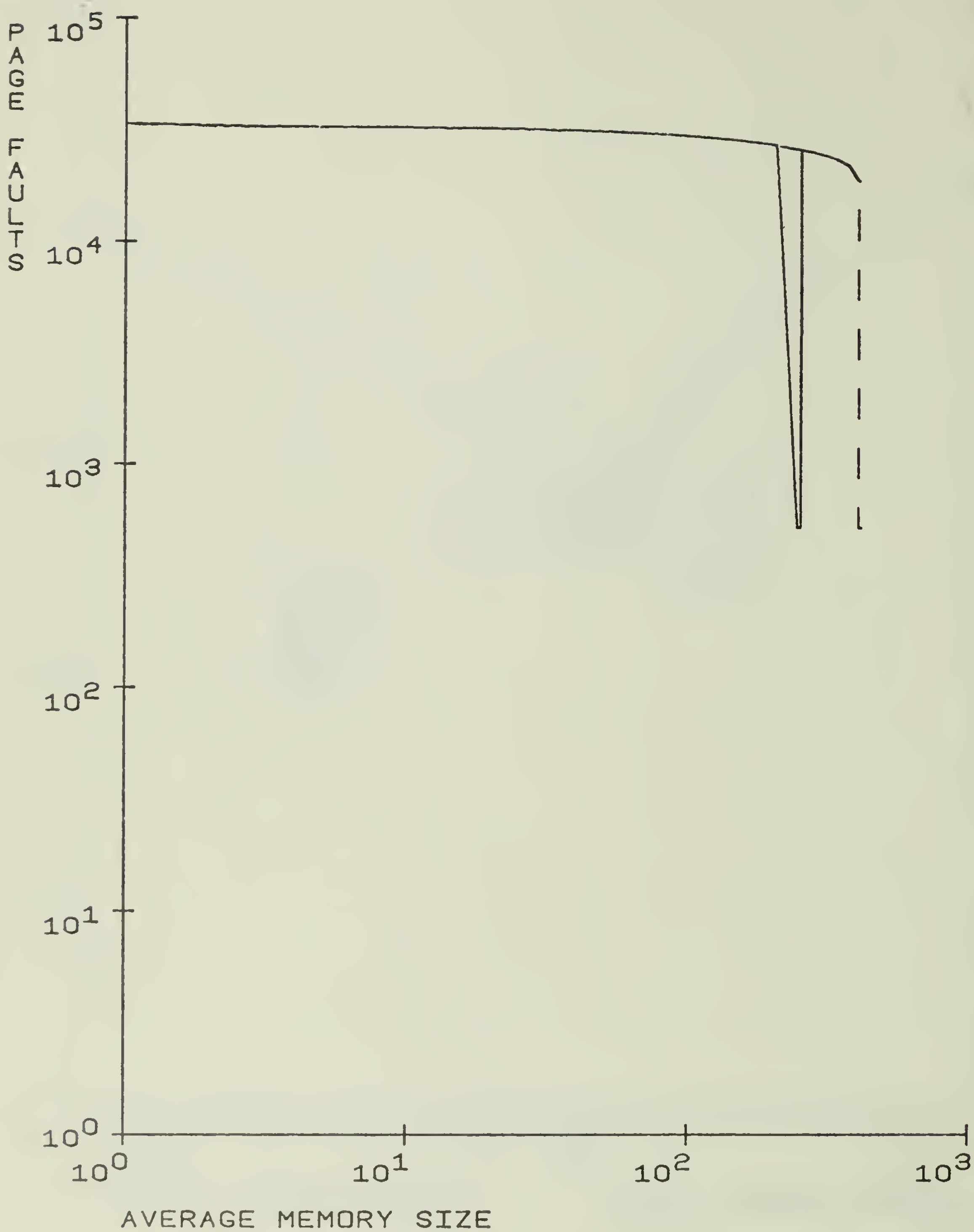

Fig. 14(a). The Page Faults vs. Average Memory Curves for Program PAPUAL (Array References)

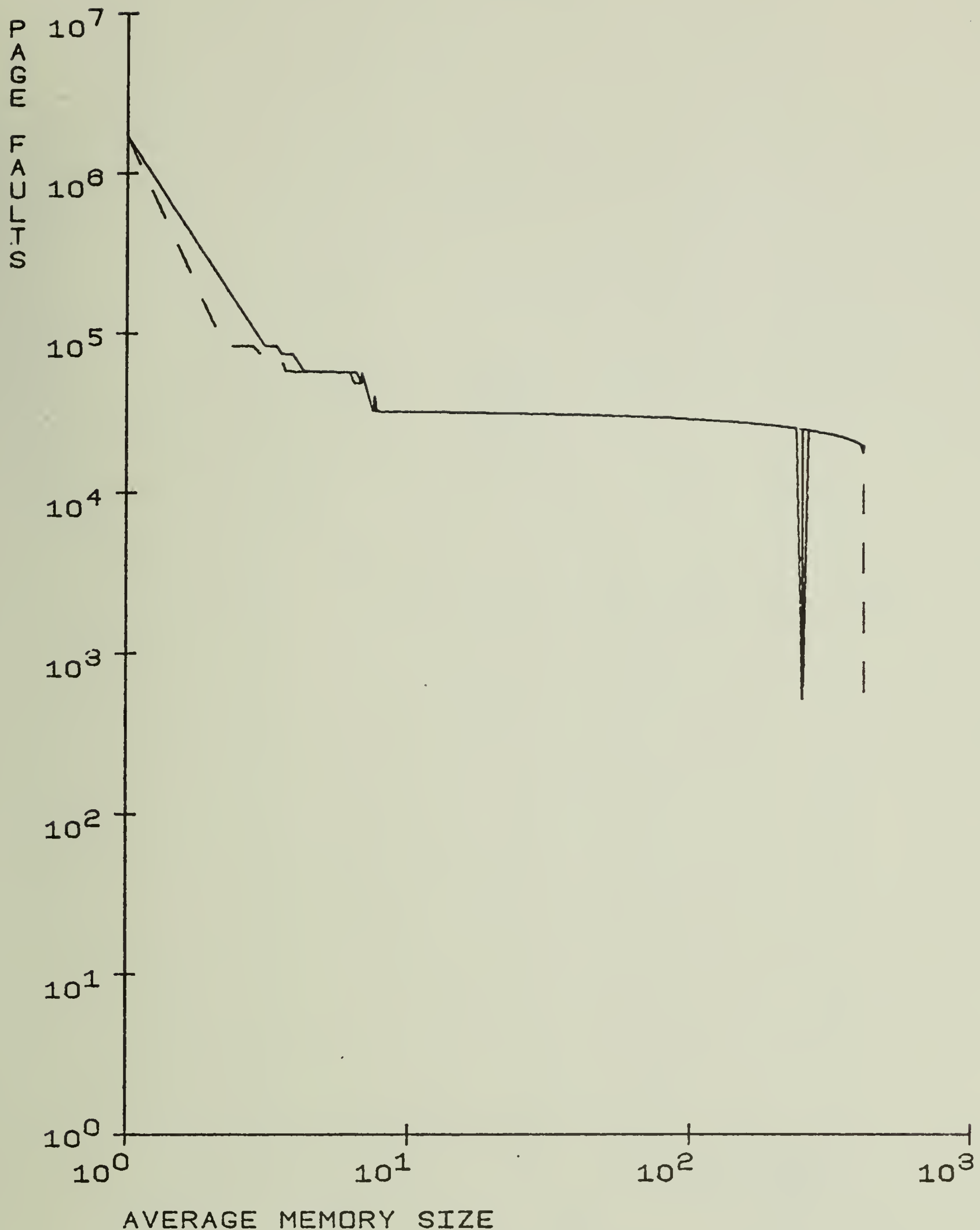

Fig. 14(b). The Page Faults vs. Average Memory Curves for Program PAPUAL (All References)

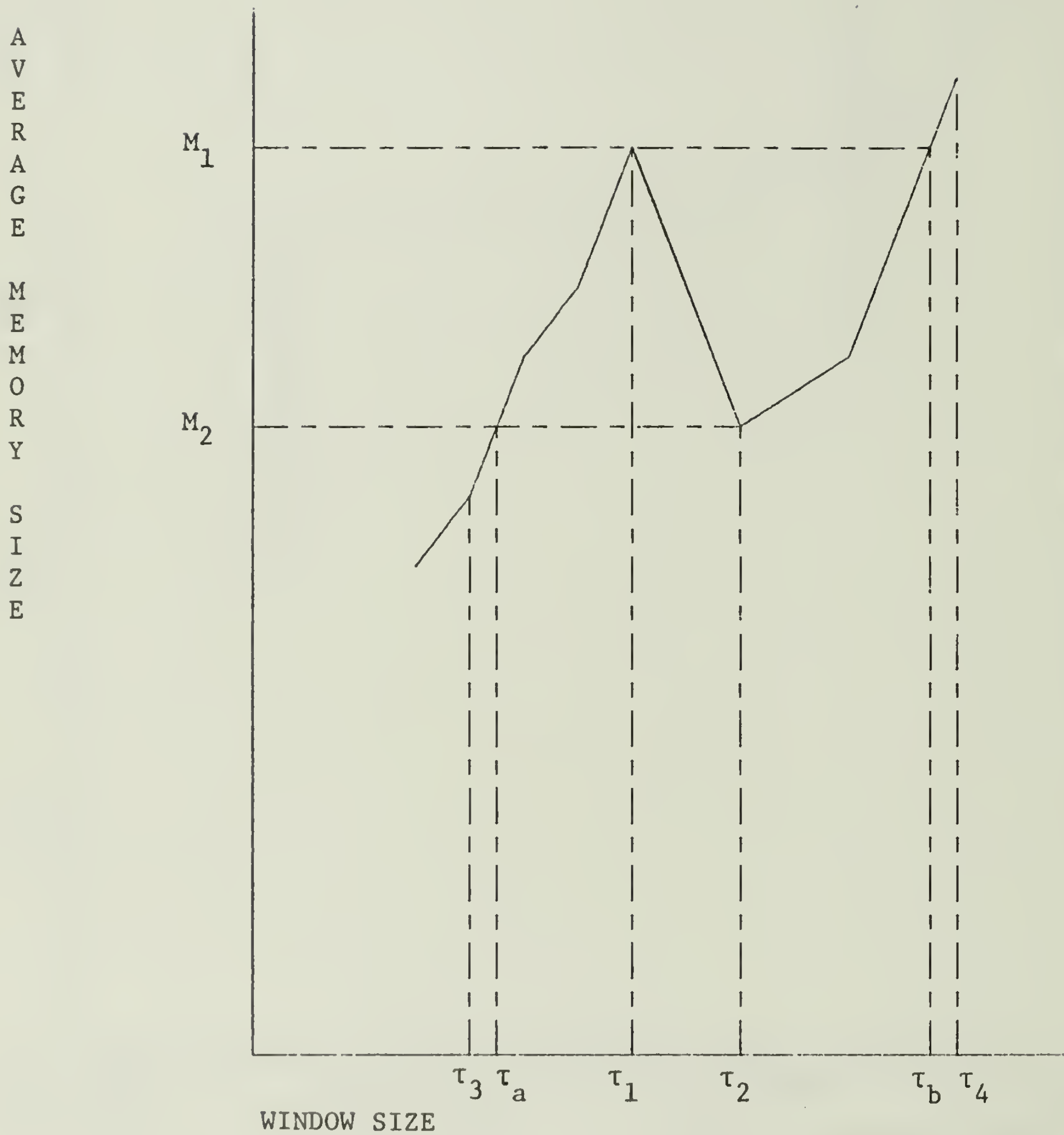

Fig. 15. Section of a Memory Allotment Curve Showing the Parameter-Real Memory Anomaly

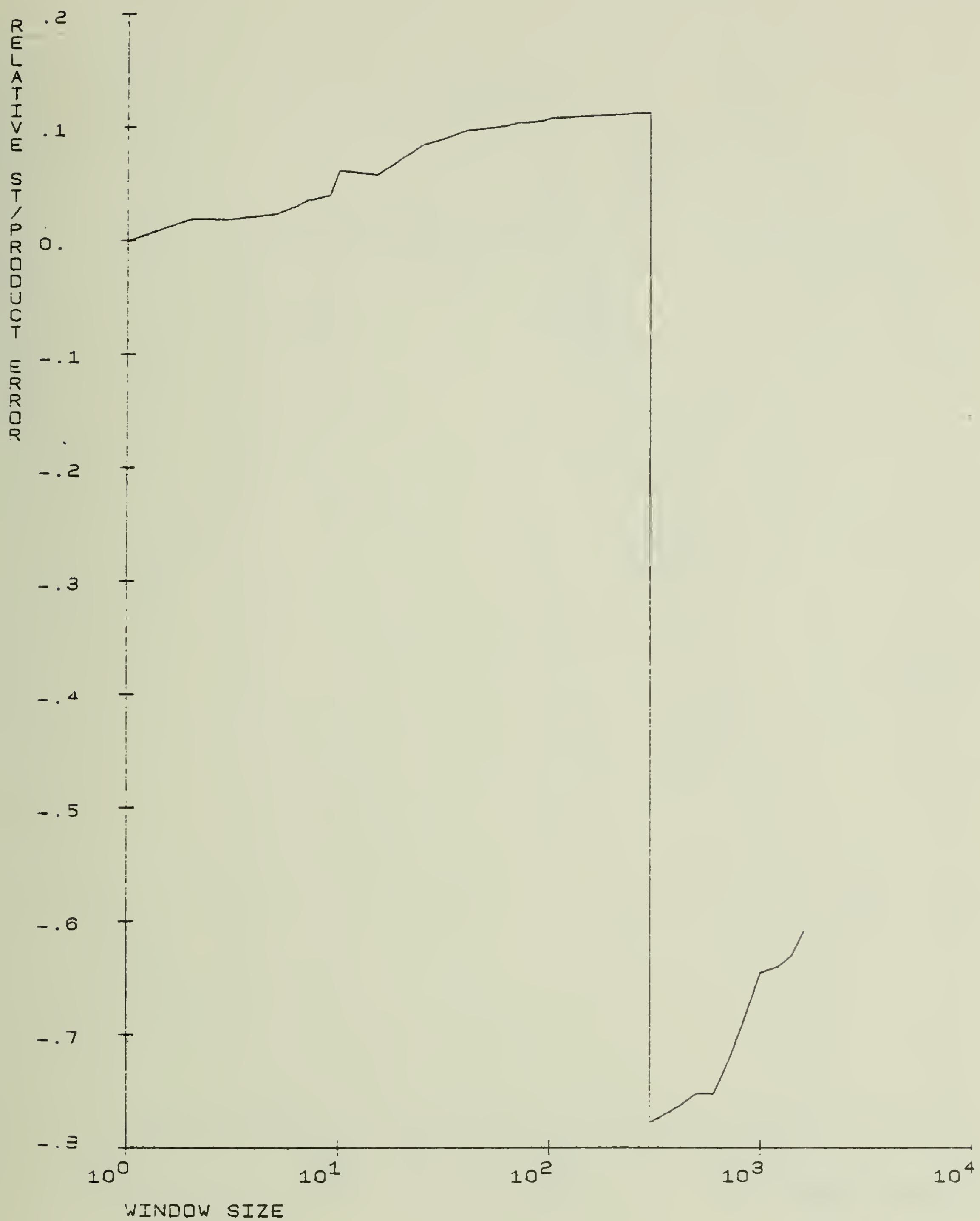

Fig. 16(a). Space-Time Product Relative Error Curves for Program BASE (Array References)

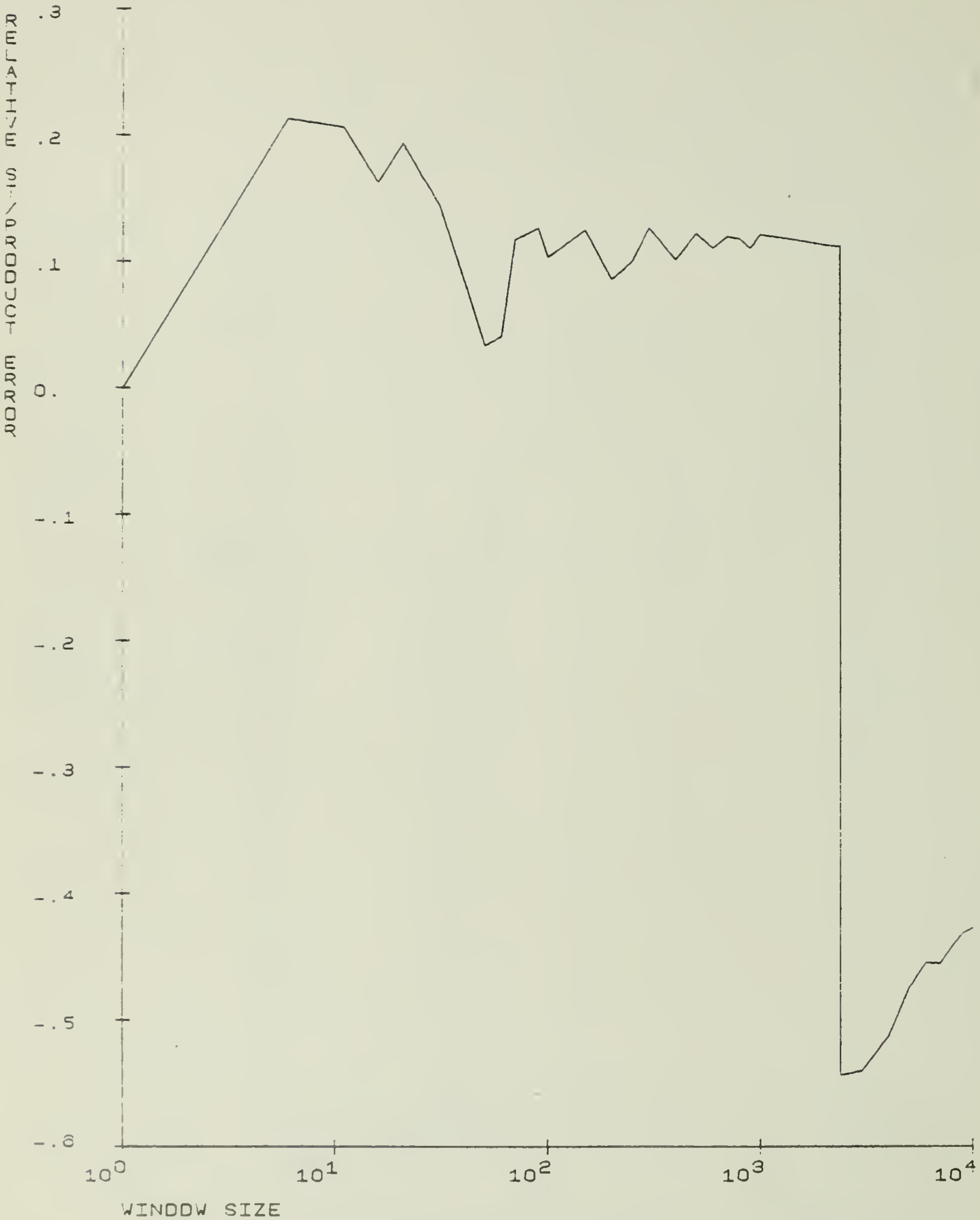

Fig. 16(b). Space-Time Product Relative Error Curves for Program BASE (All References)

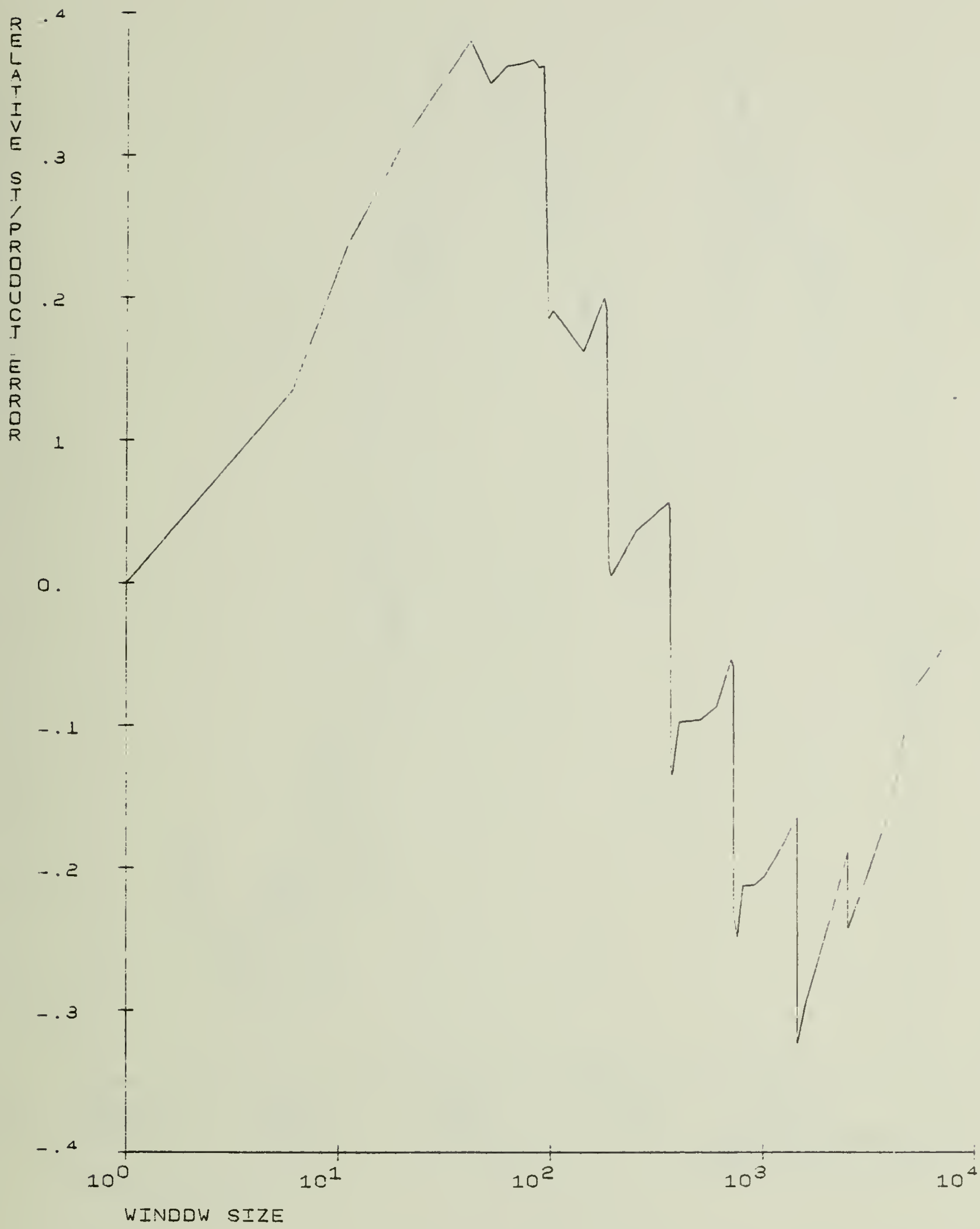

Fig. 17(a). Space-Time Product Relative Error Curves for Program FOURTR (Array References)

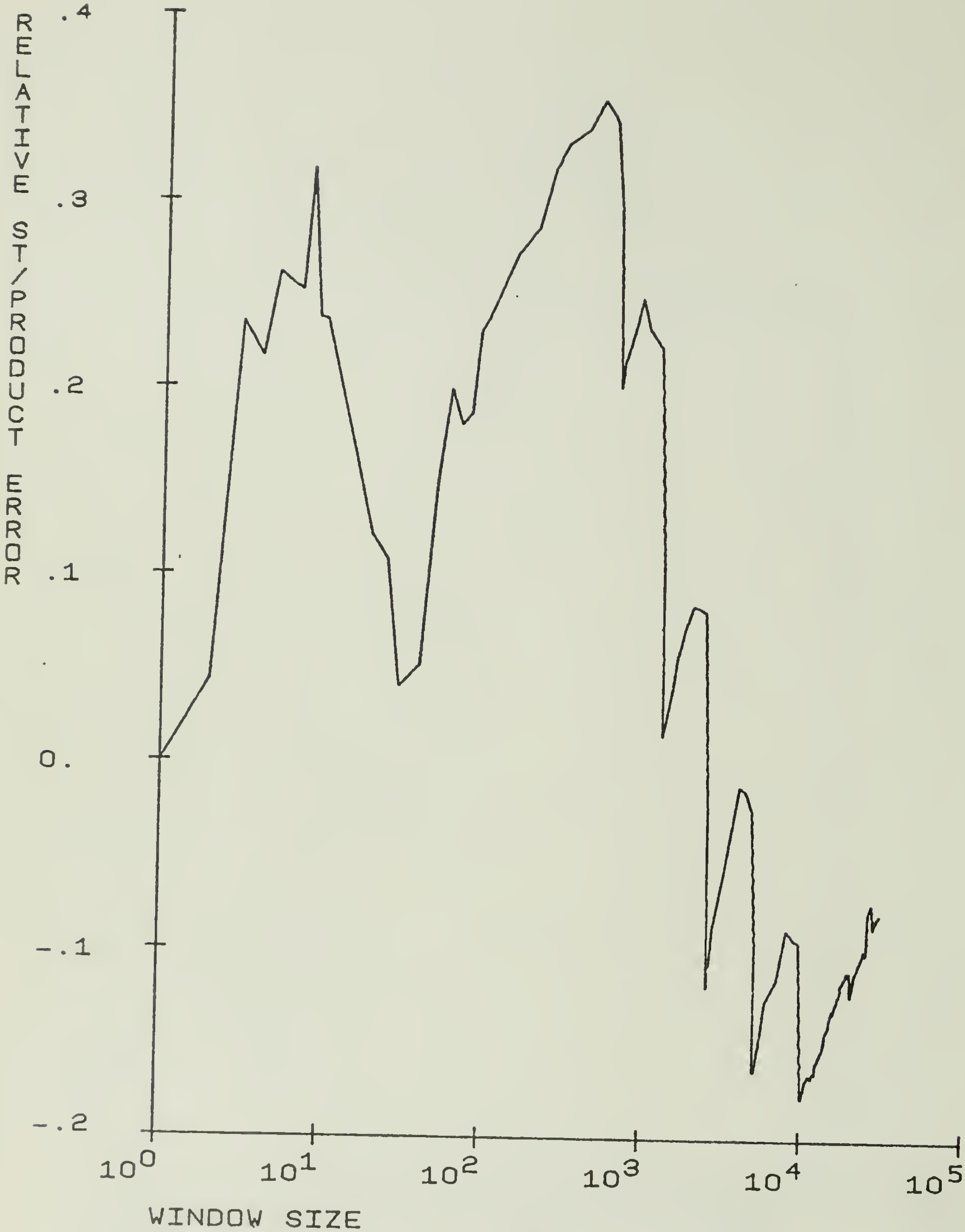

Fig. 17(b). Space-Time Produce Relative Error Curves for Program FOURTR (All References)

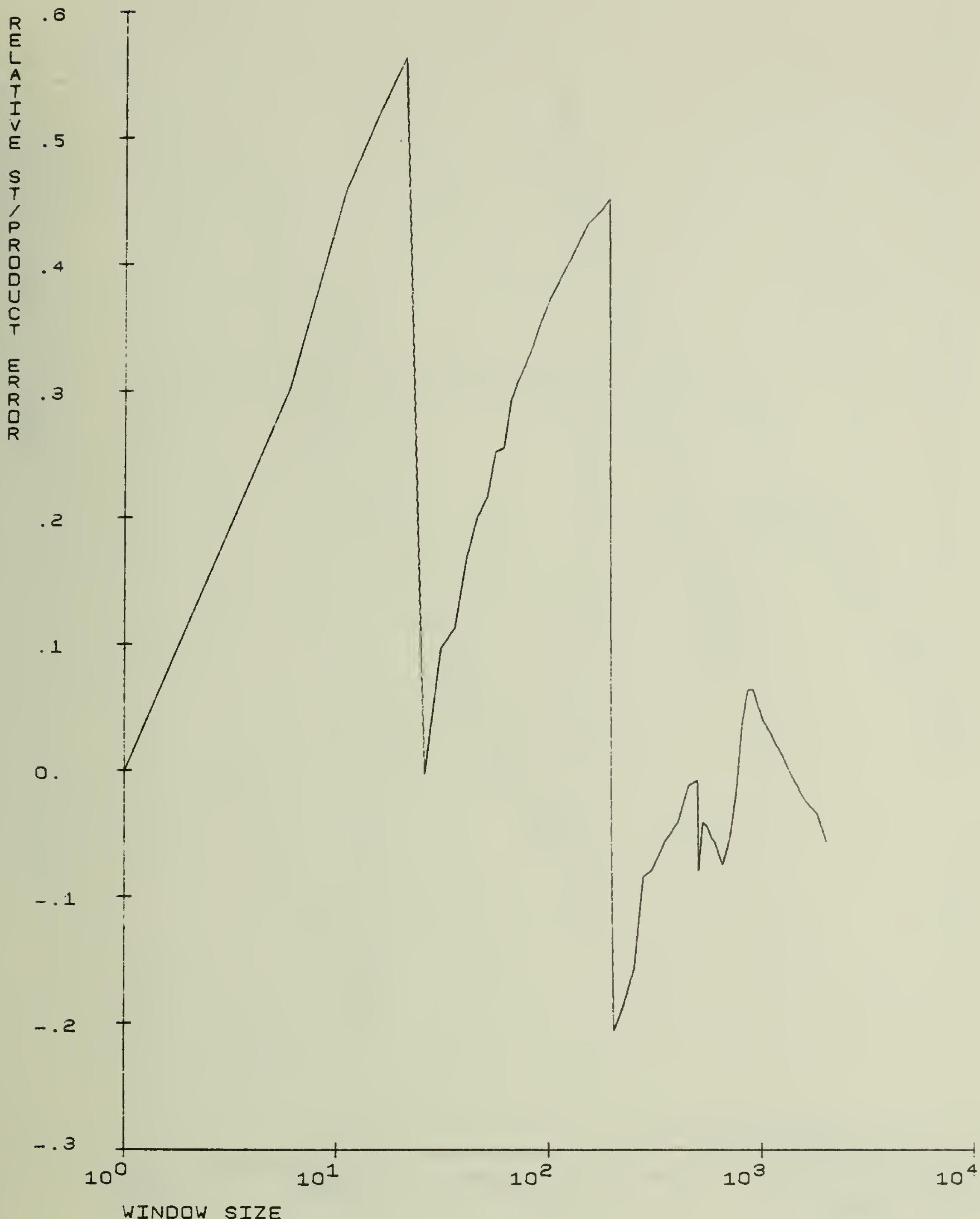

Fig. 18(a). Space-Time Product Relative Error Curves for Program INIT (Array References)

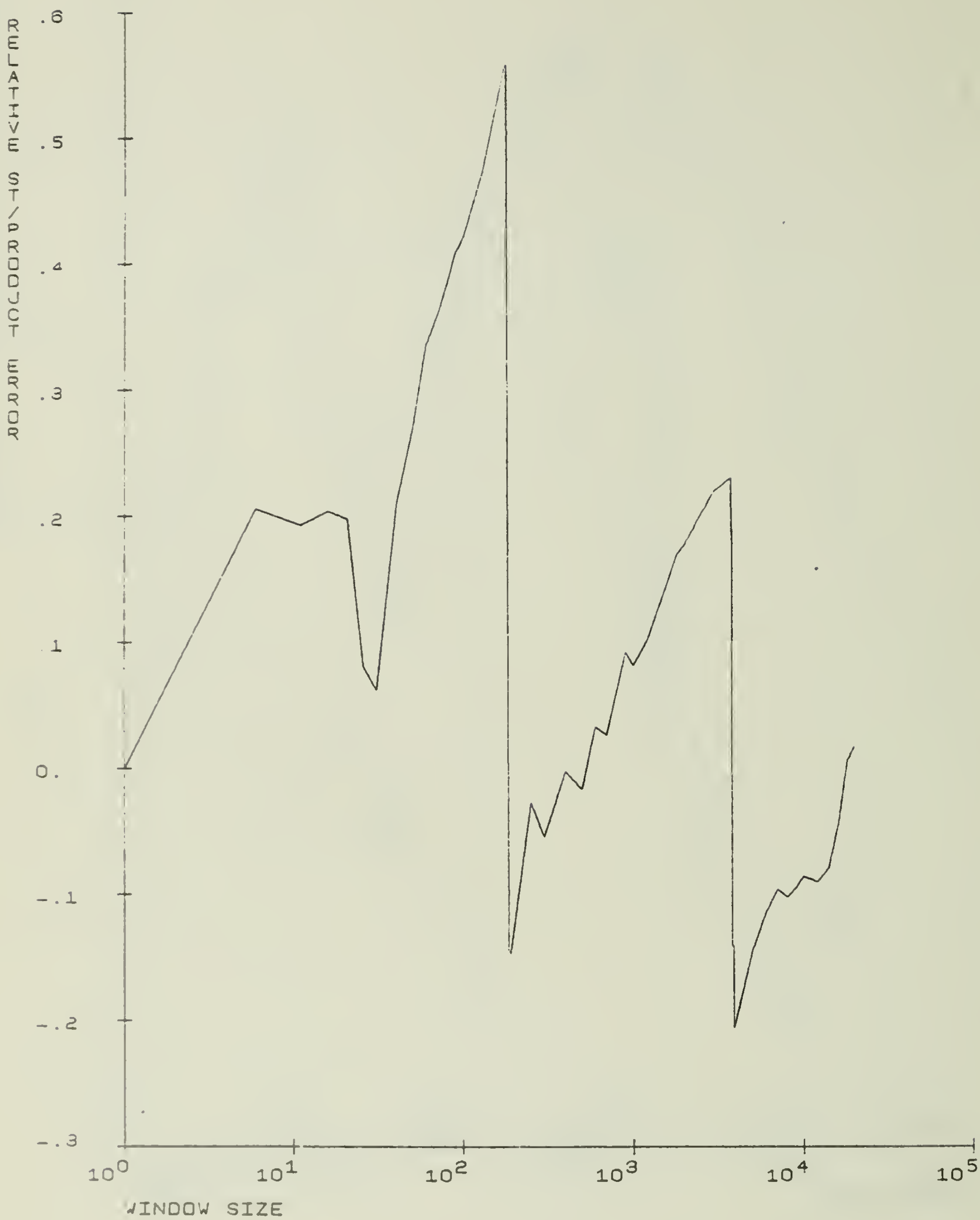

Fig. 18(b). Space-Time Product Relative Error Curves for Program INIT (All References)

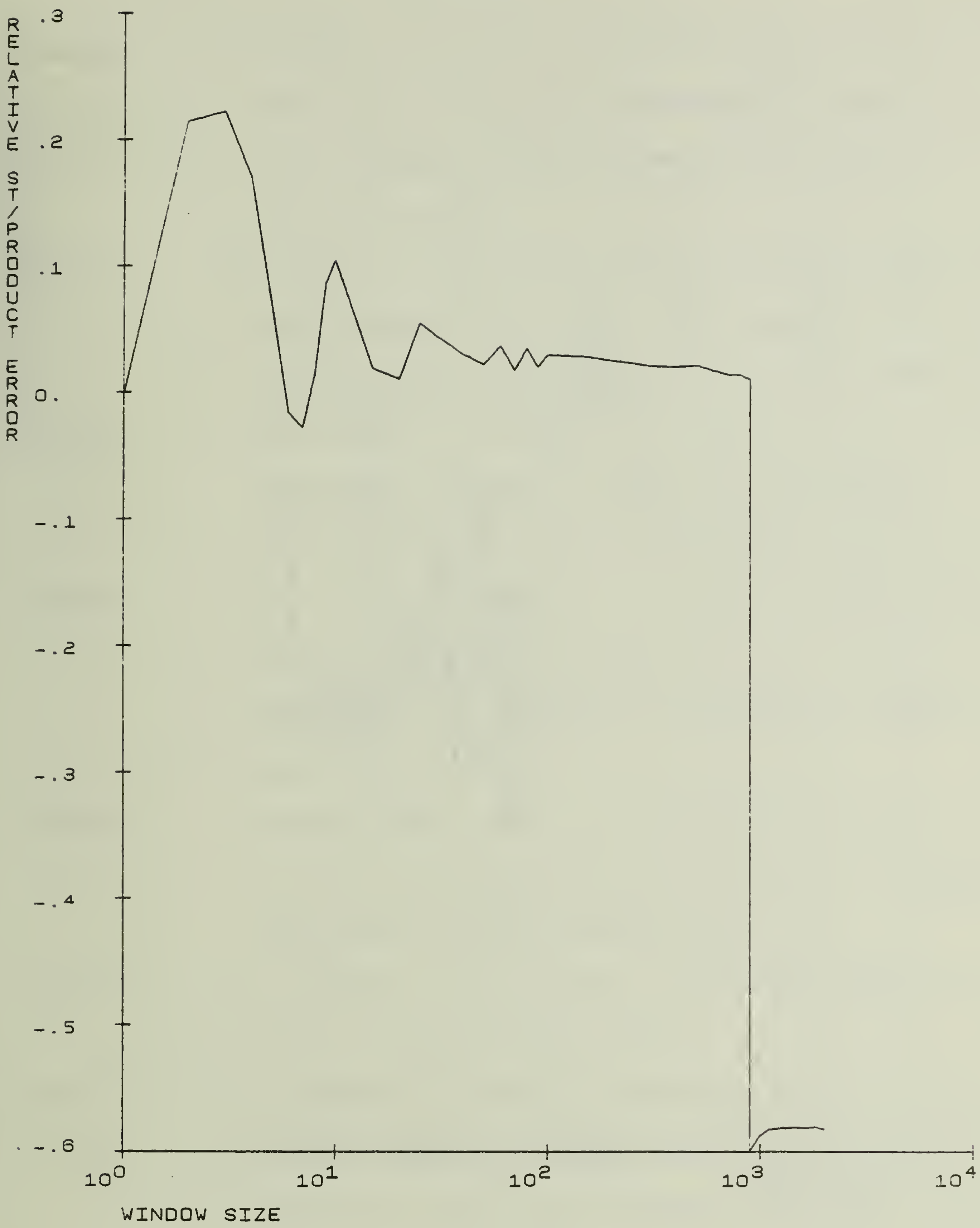

Fig. 19(a). Space-Time Product Relative Error Curves for Program PAPUAL (Array References)

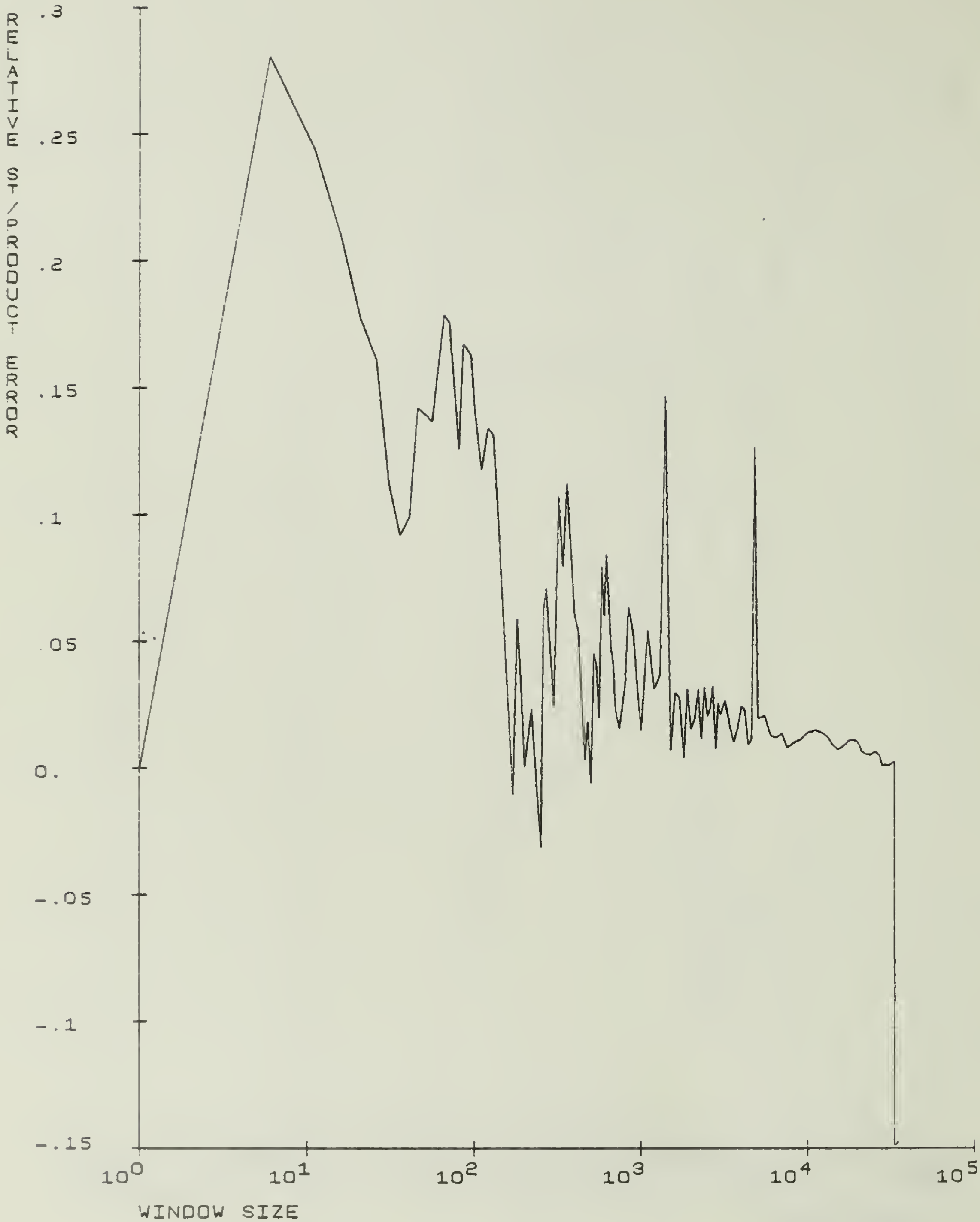

Fig. 19(b). Space-Time Product Relative Error Curves for Program PAPUAL (All References)

References

[AbuS78] W. Abu-Sufah, "Improving the Performance of Virtual Memory Computers," Ph.D. thesis, University of Illinois at Urbana-Champaign, Dept. of Computer Science Rpt. No. 78-945, Nov. 1978.

[AbKL79] W. Abu-Sufah, D. Kuck and D. Lawrie, "Automatic Program Transformations for Virtual Memory Computers," Proc. of the 1979 Nat'l. Computer Conf., pp. 969-974, June 1979.

[AbKL80] W. Abu-Sufah, D. J. Kuck and D. H. Lawrie, "On the Performance Enhancement of Paging Systems through Program Analysis and Transformations," to appear in IEEE Trans. on Computers, 1981.

[BDSM80] R. Budzinski, E. Davidson, H. Stone and W. Mayeda, "DMIN--An Optimal Algorithm for Dynamic Allocation in a Virtual Memory Computer," to appear in IEEE Trans. on Software Engineering, 1980.

[BuDa80] R. Budzinski and E. Davidson, "A Comparison of Dynamic and Static Virtual Memory Allocation Algorithm," to appear in IEEE Trans. on Software Engineering, 1980.

[Denn70] P. J. Denning, "Virtual Memory," Computing Surveys, Vol. 2, No. 3, pp. 153-189, Sept. 1970.

[Denn78] P. J. Denning, "Optimal Multiprogrammed Memory Management," in Current Trends in Programming Methodology III, K. M. Chandy and R. Yeh, Eds., Englewood Cliffs, NJ: Prentice-Hall, pp. 298-322, 1978.

[Denn80] P. J. Denning, "Working Sets Past and Present," IEEE Trans. on Software Engineering, Vol. SE-6, No. 1, pp. 64-84, Jan. 1980.

[DKLP76] P. J. Denning, K. C. Kahn, J. Leroudier, D. Potier and R. Suri, "Optimal Multiprogramming," Acta Informatica, Vol. 7, Fasc. 2, pp. 197-216, 1976.

[FrGG78] M. A. Franklin, G. S. Graham and R. K. Gupta, "Anomalies with Variable Partition Paging Algorithms," Comm. of the ACM, Vol. 21, No. 3, pp. 232-236, Mar. 1978.

[Grah76] G. S. Graham, "A Study of Program and Memory Policy Behavior," Ph.D. thesis, Purdue Univ., Dept. of Comput. Sci., Dec. 1976.

[GrDe77] G. S. Graham and P. J. Denning, "On the Relative Controllability of Memory Policies," in Computer Performance, K. M. Chandy and M. Reiser, Eds., Amsterdam, The Netherlands: North-Holland, pp. 411-428, Aug. 1977.

[Gupt74] R. K. Gupta, "Program Reference Behavior and Dynamic Memory Management," D.Sc. thesis, Washington Univ., St. Louis, MO, Dept. of Elect. Eng., Dec. 1974.

[MaBa76] A. W. Madison and A. P. Batson, "Characteristics of Program Localities," Comm. of the ACM, Vol. 19, No. 5, pp. 285-294, May 1976.

[McCo69] A. C. McKellar and E. G. Coffman, "Organizing Matrices and Matrix Operations in Paged Memory Systems," Comm. of the ACM, Vol. 12, No. 3, pp. 153-165, Mar. 1969.

[MGST70] R. L. Mattson, J. Gecsei, D. R. Slutz and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," IBM Systs. Journ., Vol. 9, No. 2, pp. 78-117, 1970.

[PrFa76] B. G. Prieve and R. S. Fabry, "VMIN--An Optimal Variable-Space Page Replacement Algorithm," Comm. of the ACM, Vol. 19, No. 5, pp. 295-297, May 1976.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-80-1043	2.	3. Recipient's Accession No.
4. Title and Subtitle SOME RESULTS ON THE WORKING SET ANOMALIES IN NUMERICAL PROGRAMS			5. Report Date November 1980
			6.
7. Author(s) Walid A. Abu-Sufah and David A. Padua			8. Performing Organization Rept. No. UIUCDCS-R-80-1043
9. Performing Organization Name and Address University of Illinois at Urbana-Champaign Department of Computer Science Urbana, Illinois 61801			10. Project/Task/Work Unit No.
			11. Contract/Grant No. US NSF MCS76-81686
12. Sponsoring Organization Name and Address National Science Foundation Washington, D. C. 20550			13. Type of Report & Period Covered Technical Report
			14.

15. Supplementary Notes

16. Abstracts

This paper shows that the Working Set parameter-real memory and real memory-fault rate anomalies mentioned by Franklin, Graham, and Gupta in [FrGG78] do occur in traces generated by real programs. The results of the detailed investigation of this anomalous behavior in four Fortran programs are presented. In some cases, a drop of a factor of two in the average memory allotment is observed when the window size is increased. In some instances, a bigger memory allotment means an order of magnitude increase in page faults.

17. Key Words and Document Analysis. 17a. Descriptors

Memory Management
Multiprogramming
Working Set
Working Set Anomaly
Program Behavior

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
RELEASE UNLIMITED	UNCLASSIFIED	59
	20. Security Class (This Page) UNCLASSIFIED	22. Price